Trabajo Fin de Grado

Grado en Ingeniería de las tecnologías de Telecomunicación

Aplicación móvil y web para la monitorización de temperatura y humedad con sensores BLE usando IONIC y Fiware

Autor: Enrique Vallespí Gil

Tutor: María Teresa Ariza Gómez

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Aplicación móvil y web para la monitorización de temperatura y humedad con sensores BLE usando IONIC y Fiware

Autor:

Enrique Vallespí Gil

Tutor:

María Teresa Ariza Gómez

Profesor titular

Dpto. de Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Aplicación móvil y web para la monitorización de temperatura y humedad con sensores BLE usando IONIC y Fiware.

Autor: Enrique Vallespí Gil

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de

Sevilla, 2020

El Secretario del Tribunal

Agradecimientos

Quisiera agradecer el trato recibido por mi tutora , Mª Teresa Ariza Gómez, que ha sido siempre muy cercano desde que empecé a asistir a sus clases de Sistemas Operativos. Su ayuda ha sido esencial en mi desarrollo académico.

A mi familia, que desde pequeño he comprobado que a través del estudio y el interés en el trabajo las cosas suceden y nada está al azar.

Y a todas las personas que me acompañan todos los días con paciencia que siempre me aportan momentos en mi vida.

Resumen

Gracias a los avances en el denominado “Internet of things” y en el campo del desarrollo de aplicaciones, es posible el desarrollo de sistemas sencillos que ayuden en la sociedad en multitud de tareas de forma sencilla y económica.

En este proyecto se ha desarrollado un sistema de recolección de datos de temperatura y humedad para sistemas de regadíos, que permite la gestión de las diferentes fuentes de información de manera sencilla para finalmente, representar los datos obtenidos en una página web.

Se ha pretendido desarrollar únicamente la solución a través de javaScript y un superconjunto de éste llamado TypeScript. Esta decisión es fundamental para abaratar costes de desarrollo y permite, entre otras cosas, la creación de aplicaciones híbridas para dispositivos móviles a través del framework Ionic, el desarrollo de un dispositivo que utiliza tecnología Bluetooth Low Energy y una página web que obtiene los datos recolectados para mostrarlos por pantalla.

Para la gestión de la información de manera centralizada se ha utilizado la tecnología Orion Context Broker y Cygnus de Fiware que confiere escalabilidad, en cuanto al número de sensores BLE, al sistema desarrollado.

Abstract

Thanks to advances in the so-called "Internet of Things" and in the field of application development, it is possible to develop simple systems that help society in a multitude of tasks in simple and inexpensive ways.

In this project, a temperature and humidity data collection system has been developed for irrigation systems, which allows the management of the different sources of information in a simple way to finally represent the data obtained on a web page.

We have tried to develop the solution only through javaScript and a superset of it called TypeScript. This decision is fundamental to lower development costs and allows the creation of hybrid applications for mobile devices through the Ionic framework, the development of a device that uses Bluetooth Low Energy technology for its management and a website that obtains the data collected and shows them on screen.
For the information management in a centralized way, the Orion Context Broker and Cygnus technology from Fiware has been used, which confers scalability regarding the number of BLE sensors to the developed system.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Acrónimos

API	Application programming interface.
BBDD	Base de datos.
BLE	Bluetooth Low Energy.
CSS	Cascading Style Sheets.
GAP	Generic Access Profile.
GATT	Generic Attribute Profile.
HTML	HyperText Markup Language.
IoT	Internet of things.
IP	Internet Protocol
NPM	Node Package Manager.
OCB	Orion Context Broker.
REST	Representational state transfer.
SDK	Software Development Kit.
UUID	Universally unique identifier.

1. INTRODUCCIÓN

El desarrollo del trabajo pretende aumentar y facilitar la producción en sistemas de cultivos. Para ello se elabora una arquitectura de un sistema que propone la recolección de datos a través de sensores de temperatura y humedad para cultivos.

Interesa que la red de sensores sea fácilmente ampliable y cómodo para el personal de trabajo, y que se complemente con un sistema de automatización que permita en casos críticos mantener los niveles de humedad y temperatura en un rango óptimo.

1.1. Motivación

El mayor interés en el trabajo es la creación de un sistema completo y funcional que responda a unas necesidades particulares utilizando en todo momento las últimas tecnologías. El uso de éstas es un requerimiento indispensable que confiere nuevos conocimientos de gran utilidad para futuros proyectos como pueden ser una introducción al Internet de las cosas (IoT) u otra utilizada diariamente, Bluetooth Low Energy (BLE) entre otras.

Cabe destacar el uso de la tecnología Docker para la puesta en marcha de forma rápida y sencilla.

1.2. Objetivos

El objetivo principal del proyecto es almacenar datos de temperatura y humedad obtenidos a través de los sensores usados para su procesamiento y representación en una página web.

Se puede implementar la red de sensores tanto en sistemas de cultivos menores como una huerta propia, hasta otros más extensos como pueden ser los cultivos de invernaderos en Almería.

Para ello se elabora una interfaz de usuario a través de una aplicación móvil que permita la gestión de los sensores. Esta interfaz permite activar y desactivar los

sensores individualmente.

Por último, con el fin de permitir la escalabilidad del sistema, el procesamiento de datos se realiza a través del manejador de contexto de Fiware.

1.3. Antecedentes

Al entrar en el estudio del grado, y tras cursar los años aprendiendo las bases teóricas y desarrollando tareas con fines académicos, quería realizar un trabajo con finalidad tangible que me alentara a seguir aprendiendo cada vez conocimientos más técnicos.

He utilizado los recursos obtenidos durante estos años aplicándolos en la solución final de manera indirecta. Sin estos conocimientos previos hubiese sido inviable la materialización de la solución.

1.4. Descripción de la solución

Se ha desarrollado una aplicación completa cuyos componentes son:

- Aplicación de recolección de datos: utiliza un sensor AM2302 conectado a una Raspberry Pi. Envía los datos de temperatura y humedad de forma periódica a un nodo central. Ofrece una interfaz de gestión que permite modificar parámetros como el nombre del dispositivo o su modo de funcionamiento a través de la tecnología BLE. El lenguaje de programación utilizado para el desarrollo de la aplicación ha sido JavaScript.
- Aplicación móvil: se ha desarrollado una aplicación móvil que funciona como interfaz de usuario del sensor. Permite iniciar o finalizar la transferencia de datos de temperatura y humedad entre la raspberry pi y el servidor principal, así como modificar el nombre del sensor seleccionado para identificar dichos datos representados en la aplicación web. Para la comunicación entre el sensor BLE y la aplicación móvil se ha utilizado la tecnología BLE. Para el desarrollo de la aplicación hemos elegido el framework Ionic y el lenguaje de programación javascript.
- Aplicación web: publica los datos obtenidos de forma independiente para cada sensor. Se muestran en dos gráficas distintas facilitando su consulta a través de un filtro de rango de fechas, y permite al usuario seleccionar el sensor a consultar. Está desarrollado en Angular en conjunción con NodeJs

y MySQL.

- Manejador de contexto (context broker): con el fin de aportar al sistema escalabilidad se propone el uso de un manejador de contexto que recopile la información a través de entidades de contexto. Facilita su recepción y envío entre sumideros y fuentes de datos. Se ha utilizado la tecnología Orion Context Broker de Fiware.

1.4.1. Esquema de la arquitectura

En la figura 1 se observa cómo se conectan los recursos hardware utilizados en el proyecto con dos sensores y un único Smartphone. Como se indica anteriormente se ha desarrollado para ser escalable.

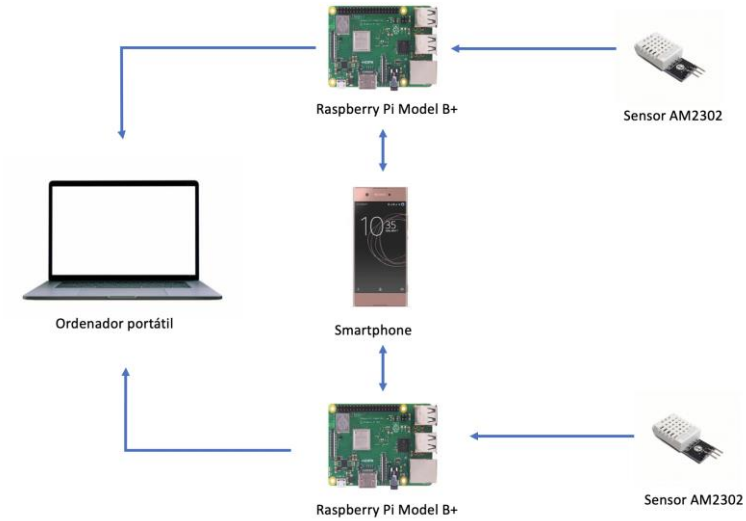

Figura 1: Recursos hardware utilizados

En la figura 2 se representa el mismo escenario, detallando las diferentes aplicaciones desarrolladas e indicando en la leyenda la tecnología aplicada en cada elemento.

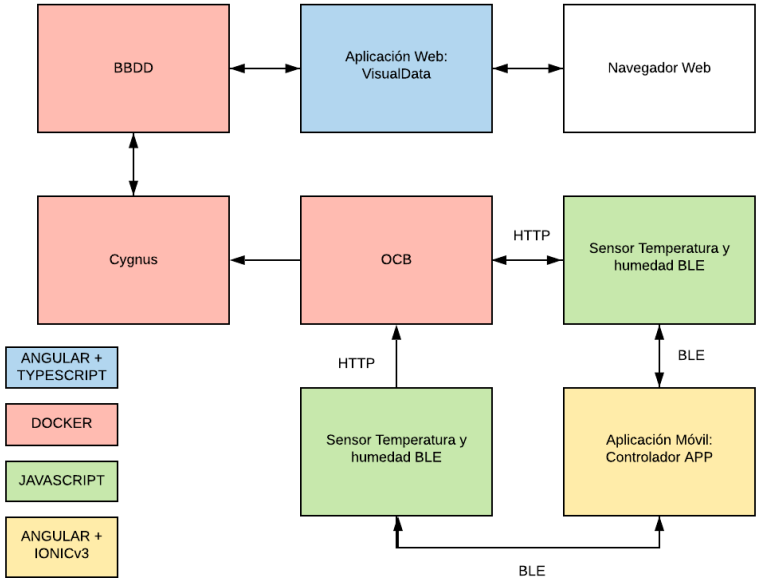

Figura 2: Tecnologías aplicadas

1.5. Estructura de la memoria

Se expone a continuación los principales apartados que componen la memoria:

- Necesidades relacionadas con el hardware y software.
- Enumeración y explicación de las diferentes tecnologías utilizadas.
- Estudio funcional de la aplicación desarrollada.
- Explicación de diferentes aspectos del proyecto, como la conexión del sensor con la raspberry pi o la instalación de Frameworks utilizados entre otros.

2. RECURSOS UTILIZADOS

En este apartado se detallan los diferentes recursos utilizados en el sistema dividiéndolos entre recursos hardware y software.

2.1. Recursos Hardware

2.1.1. Ordenador de escritorio Intel Core i7-4710MQ

Para la realización del proyecto se ha utilizado un ordenador de sobremesa Intel Core i7-4710MQ, donde se aloja la aplicación web como, el OCB y la BBDD. Sus características son:

- Procesador: Intel(R) Core™ i7-4710MQ CPU @ 2.50GHz, 2501 MHz, 4 procesadores principales, 8 procesadores lógicos.
- Placa Base: Notebook W35xSS_370SS
- Memoria RAM: 16 GB.
- Almacenamiento: 1TB HHD + 256 GB SSD.
- Sistema Operativo: Windows 10 Pro.

Las necesidades mínimas de memoria RAM, almacenamiento y Procesador son menores a las expuestas para el correcto funcionamiento del OCB y la aplicación web.

2.1.2. Raspberry Pi Model B+

Utilizado para alojar el software necesario relacionado con el sensor BLE. Se conecta con el sensor a través de los pines GPIOs (se detalla en el Anexo A).

Se ha elegido este modelo debido a que tiene un coste bajo en relación a la gran capacidad computacional que ofrece.

Permite de forma más sencilla el desarrollo del dispositivo al contar con un módulo Bluetooth y Wifi de fábrica.

Además, su bajo consumo y su pequeño volumen lo hace muy interesante en implementaciones similares.

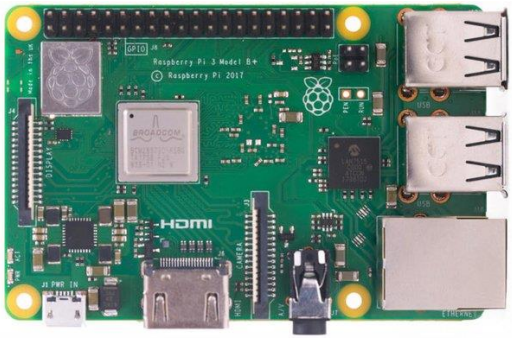

Figura 3: Raspberry Pi Model B+

Sus características son:

- Procesador: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit 1.4GHz
- Memoria RAM: 1GB
- Adaptador Wi-Fi: 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac
- Adaptador BLE
- Puerto Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)
- GPIO: 40 pines
- Micro-SD

2.1.3. Sensor AM2302

Obtiene datos de humedad y temperatura con un consumo bajo, precio reducido y buena preecisión en las muestras. Al ser un periférico comúnmente utilizado, existen numerosos módulos que permiten su utilización, en nuestro caso se ha utilizado la librería 'node-dht-sensor' incluídas en NPM.

Sus características son:

- Alimentación: 3.3Vdc ≤ Vcc ≤ 6Vdc
- Rango de temperatura: -40°C a 80 °C
- Precisión de temperatura: <±0.5 °C
- Resolución temperatura: 0.1°C
- Rango de humedad: De 0 a 100% RH
- Precisión de humedad: 2% RH
- Resolución humedad: 0.1% RH
- Tiempo de muestreo mínimo: 2s

Figura 4: Sensor AM2302

2.1.4. Dispositivo móvil Sony Xperia G3121

Encargado de comunicarse con el sensor a través de la aplicación móvil desarrollada. Implementa la interfaz de usuario.

Sus características son:

- Procesador: MediaTek Helio P20 2.3GHz.
- Memoria RAM: 3 GB.
- Almacenamiento: 32GB.
- Sistema Operativo: Android 7.0.

Figura 5: Sony Xperia G3121

2.2. Recursos Software

2.2.1. Visual Studio Code

Potente editor de código fuente, abierto y gratuito, desarrollado y lanzado por Microsoft en 2015, compatible con multitud de lenguajes de programación diferentes (Windows, MacOS y Linux). Viene con soporte incorporado para JavaScript, TypeScript y Node.js, y posee multitud de extensiones para otros lenguajes (C++, C#, Java, Python, PHP, GO)[1]

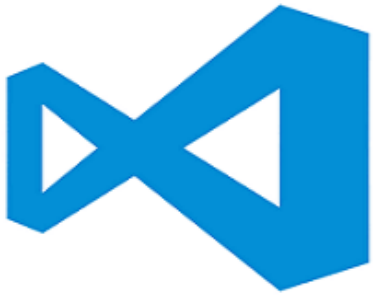

Figura 6: Logo Visual Studio Code

2.2.2. Android Studio

Entorno de desarrolo integrado (IDE) oficial para el desarrollo de apps para Android.[2]. Desarrollado por Google en 2013, reemplazó a Eclipse como el IDE oficial.

Su relevancia en el proyecto se centra únicamente en el proceso de depuración de la aplicación móvil y al ser ésta híbrida su uso es completamente opcional.

Figura 7: Logo Android Studio

2.2.3. MySQL Workbench

Es una herramienta con interfaz gráfica que permite el desarrollo de bases de datos, así como su administración y gestión. Está disponible en Windows, Linux y Mac OS

X. [3]

Ha sido de gran relevancia para entender el funcionamiento de Cygnus trabajando con OCB. Además, nos ha permitido desarrollar las peticiones a la BBDD desde el Back End de la aplicación web de manera más sencilla.

Figura 8: Logo MySQL Workbench

2.2.4. Postman

Herramienta utilizada mayoritariamente para el testeo de API REST. Además, cuenta con otras funcionalidades como la posibilidad de automatizar pruebas y creación de mockups.

Empezó como una extensión de Google Chrome. Actualmente tiene soporte para Windows, Macs y Linux.

Tiene la posibilidad de almacenar las diferentes pruebas realizadas en la nube para compartirla con el equipo de trabajo.

Figura 9: Logo Postman

2.2.5. LightBlue

Herramienta utilizada durante el desarrollo de aplicaciones que utiliza tecnología BLE. Permite al dispositivo conectarse con un periférico BLE. Probando los servicios y características.

Además, permite convertir al dispositivo móvil en un periférico BLE pudiendo configurar todas sus características. Las diferentes configuraciones se pueden exportar e importar en otros dispositivos que utilicen dicha aplicación.

En nuestro caso lo hemos utilizado para probar el funcionamiento de los sensores BLE sin la necesidad de tener la aplicación móvil desarrollada.

Figura 10: Logo LighBlue

3. TECNOLOGÍAS UTILIZADAS

En este apartado indicamos las diferentes tecnologías utilizadas en el desarrollo del proyecto.

3.1. Angular

Es un Framework de código abierto desarrollado por Google para aplicaciones web desarrolladas en TypeScript. Utiliza el Modelo Vista Controlador y es utilizado para el desarrollo de aplicaciones web de una sola página.

Angular se basa en el uso de componentes, que son pequeñas partes lógicas que tiene su representación en pantalla. Estos se dividen en tres archivos de extensión .HTML, .css y .ts (TypeScript), y se exportan a través de clases que pueden ser importadas en otros componentes.

Existen, además de los componentes, servicios que son clases TypeScript y contienen entre otros la lógica que permiten acceder a BBDD. Estos servicios se inyectan en los componentes y pueden ser utilizados como un nuevo objeto con sus propios métodos[4].

Figura 11: Logo Angular

3.1.1. BootStrap

Framework gratuito para el desarrollo del Front End en aplicaciones web. Incluye plantillas de HTML y CSS que permiten desarrollar de manera más eficiente y rápida.

Contiene un gran número de componentes que son fácilmente implementados en el

proyecto angular desarrollado[5].

Figura 12: Logo BootStrap

3.2. NodeJs

Es un "entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome"[6].

Permite el uso de JavaScript para el Back End de una aplicación. Al estar basado en eventos, se consigue que sea eficiente y liviano permitiendo su utilización en dispositivos IoT.

Figura 13: Logo NodeJs

En principio, su objetivo principal era poder tener 10.000 conexiones concurrentes en el lado de servidor de manera sencilla. La diferencia fundamental con otras tecnologías de servidor es que escucha todas las peticiones de clientes en un solo hilo posibilitando su ejecución de manera más veloz.

Actualmente está muy respaldado por la comunidad de desarrolladores, siendo utilizada por LinkedIn, Uber o eBay entre otros.

Además del auge que está teniendo NodeJs, hemos elegido dicha tecnología debido a su versatilidad, pues se ha desarrollado tanto el servidor web como el software

ejecutado en la raspberry pi para funcionar como sensor BLE.

Gracias a su gestor de paquetes NPM su instalación y administración resulta muy sencilla.

3.2.1. Gestor de paquetes NPM

Npm es el sistema gestor de paquete de Node considerado el ecosistema de librerías más grande en la actualidad. Permite de manera sencilla instalar librerías, administrar las dependencias del Proyecto y distribuir el código.

Figura 14: Logo NPM

3.3. IONIC Framework

Ionic Framework es un SDK de código abierto para el desarrollo de aplicaciones móviles híbridas.

Las aplicaciones híbridas están construidas a través de HTML5, Js y CSS y permiten el desarrollo, a través de un único código fuente, de aplicaciones para diversas plataformas.

Existen numerosos FrameWorks que permiten el desarrollo de aplicaciones híbridas. En nuestro caso hemos utilizado IONIC debido a que es código abierto y además tiene una gran comunidad de desarrolladores.

La versión 3 del framework, utilizada en este proyecto, utiliza Angular y Cordova, aunque actualmente puede trabajar con ReactJs y VueJs. Además, han desarrollado Capacitor (disponible en la versión 4 del framework), que es la implementación propia de Apache Cordova, consiguiendo así que el rendimiento de la aplicación

desarrollada sea superior.

Se ha utilizado el FrameWork para la realización de la interfaz de usuario que controla los sensores a través de la aplicación móvil desarrollada [7].

Figura 15: Logo IONIC Framework

3.3.1. Apache Cordova

Herramienta opensource desarrollada para la creación de aplicaciones móviles a través de HTML, CSS y JavaScript. Es la versión de código abierto de PhoneGap.

Actualmente es usada por multitud de frameworks como Mónaco, Taco y Telerik.

Figura 16: Logo Apache Cordova

3.4. Docker

Docker es un contenedor software que permite la virtualización de multitud de herramientas diferentes, incluyendo su código y todas las dependencias en una imagen. Esto confiere al producto portabilidad e inmutabilidad.

Además, la tecnología Docker utiliza parte del Sistema Operativo anfitrión para conferirle ligereza, pudiéndose ejecutar en multitud de máquinas con S.O. diferentes.

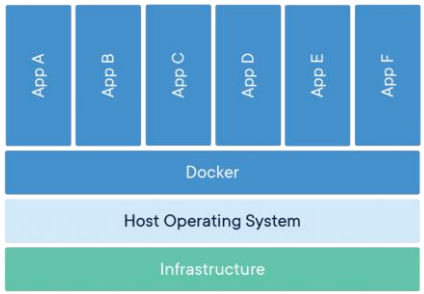

Figura 17: Aplicacionnes contenedores

Hemos utilizado la herramienta "Docker-Compose" para definir y arrancar todas las instancias con solo un comando. Esta tecnología utiliza un fichero tipo YAM para la definición de las instancias y cómo se relacionan entre ellas.

Se ha utilizado la tecnología Docker para alojar el OCB, Cygnus y la BBDD MySQL.

En el ANEXO B explicamos cómo descargar, instalar y levantar las instancias necesarias para el proyecto expuesto[8].

Figura 18: Logo Docker

3.5. Fiware Orion Context Broker

Financiado por el programa I + D dentro de la Unión Europea, Fiware tiene como objetivo desarrollar una plataforma orientada al desarrollo de nuevas aplicaciones con implementación directas en "Smart Cities" a través de diversas fuentes como redes de sensores[9].

Estos sensores emiten los datos recogidos al OCB. Componente principal de cualquier solución Fiware, debido a que proporciona toda la lógica necesaria para

administrar, consultar y actualizar la información de contexto. Utiliza la NGSI APIV2 para poder gestionar todo el ciclo de vida de la información de contexto.

Figura 19: Fiware Orion

Una característica a tener en cuenta es el ciclo de vida de dicha información, debido a que OCB la almacena hasta una futura actualización de su valor. Es por ello que resulta de gran interés la utilización de la tecnología Cygnus descrita en el siguiente apartado 3.5.1.

3.5.1. Cygnus

Cygnus es un componente adicional en la plataforma Fiware que permite almacenar de forma indefinida la información de contexto procesada. Para ello coopera con el OCB a través de las suscripciones[11].

Las suscripciones emiten la información del OCB a Cygnus cuando recibe una actualización en una entidad de contexto. Se apoya en plataformas de BigData o en bases de datos tradicionales.

3.6. MySQL

Es un tipo de base de datos relacional caracterizada por su facilidad en el uso y ser de código abierto.

Actualmente es una de las más usadas y tiene gran repercusión en entornos de desarrollo web.

Aporta seguridad en la conexión, integridad referencial (clave externa de una tabla de referencia siempre debe aludir a una fila válida de la tabla a la que se haga referencia) y transacciones (característica que indica la integridad de los datos a través de la no existencia de estados intermedios en las transacciones realizadas).

Figura 20: Logo MySQL Workbench

Se ha virtualizado una instancia MySQL en un contenedor Docker. Como se ha indicado en el apartado 3.5.1, dicha instancia es utilizada por Cygnus para aportar persistencia a los datos recogidos por el sensor.

3.7. Bluetooth Low Energy

Bluetooth Low Energy es un estándar de comunicación que, a diferencia de su antecesor, busca establecer un medio de comunicación con el menor consumo de energía posible provocando en consecuencia un radio de alcance menor[10].

Utiliza una menor potencia y ancho de banda durante la comunicación.

Está especialmente diseñado para una implementación mucho más sencilla permitiendo que desarrolladores con pequeños wearables puedan darle uso a esta tecnología y aplicarla en sus proyectos.

Figura 21: Logo Bluetooh Low Energy

Para entender el protocolo tenemos que definir:

- GAP: Acrónimo de Generic Access Profile, permite al dispositivo anunciarse al resto de intervinientes en una comunicación. Definir roles para los dispositivos siendo estos periféricos o centrales. Su tarea fundamental es emitir el Advertising data Payload mediante el cual emite información relevante a dispositivos centrales.

- GATT: Acrónimo de Generic Attribute Profile, define una interfaz que proporciona servicios y características desde un periférico para explotar sus datos en una comunicación dada.
- Servicios: Cada servicio es una interfaz única reconocida por su identificador UUID, que contiene una serie de características que le dan al servicio una utilidad específica. Las características almacenan datos y proporcionan métodos para su procesado.
- Característica: Unidad mínima de información. Están identificadas, como los servicios, por un UUID. Las características pueden ser de escritura o lectura.

En la figura 22 se puede comprobar una ilustración de lo explicado anteriormente sobre Servicios y característica.

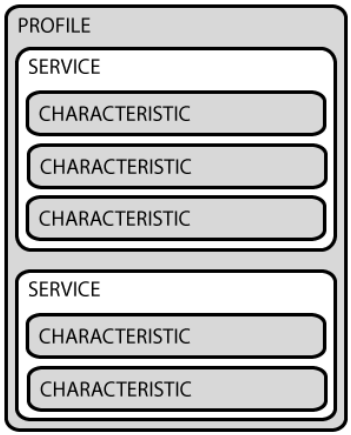

Figura 22: Servicios y características Bluetooh[12]

4. APLICACIÓN DESARROLLADA: DISPOSITIVO BLE

A continuación, se procede a describir el dispositivo BLE desarrollado en el proyecto que permite la recolección de datos de temperatura y humedad y su posterior envío a un servidor central.

4.1. Dispositivo

La implementación del dispositivo BLE se ha desarrollado a través de la tecnología NodeJS. Para ello se ha utilizado un módulo llamado Bleno.

Bleno permite de manera sencilla desarrollar un dispositivo BLE que permite la recogida de datos de temperatura y humedad, e implementa la comunicación a través de BLE con el dispositivo móvil (interfaz de control del usuario), además implementa el envío de dichos datos al OCB.

Para conseguir la escalabilidad del producto, se ha desarrollado el código que permite al dispositivo BLE almacenar todos los datos de configuración y estado del mismo, consiguiendo por tanto su trabajo de forma independiente al resto de dispositivos BLE y dispositivos de control.

4.2. Código desarrollado

El código principal desarrollado se compone de dos ficheros: Uno que permite la recolección de datos del sensor AM2302, y otro que permite el envío de las muestras a través de HTTP al OCB y la gestión del propio sensor a través de BLE.

Además, se utilizan otros ficheros secundarios que recogen constantes usadas por el dispositivo BLE, así como la definición de clases necesarias para el funcionamiento del dispositivo, como pueden ser los servicios y características, necesarios para la comunicación.

4.2.1. Funcionalidad de recolección de muestras de temperatura y humedad

Se procede a describir la funcionalidad desarrollada que permite la recolección de muestras de temperatura y humedad.

Se ha utilizado la librería "node-dht-sensor" para interactuar con el sensor conectado a la raspberry pi, y todo el código implicado en la funcionalidad se ha desarrollado en el documento "sensor.js".

Este documento exporta un módulo que define dos variables (humedad y temperatura) y una función que sobrescribe en las variables los valores obtenidos de humedad y temperatura.

Este módulo se importa en el fichero test.js, que lo utiliza para recolectar los datos de temperatura y humedad, para su posterior envío al OCB y/o a la aplicación móvil.

Para un correcto funcionamiento, se ha implementado de forma asíncrona la recolección de los valores obtenidos por el sensor, a través de las denominadas funciones flechas de JavaScript.

La función "capture", que exporta el módulo y permite a otros componentes utilizar la funcionalidad desarrollada, se expone en la figura 23.

```
const sensor = require('node-dht-sensor');
//Configuramos la conexión de la raspberry pi con el sensor
const sensorNumber = 22;
const pinNumber = 4;
//Definimos el módulo a exportar
var Sensor_data = module.exports = {
  temp: Number,
  humed: Number,
  //Definimos la función que obtiene las muestras
  capture: function(callback){
      //Utilizamos la función del modulo importado
      sensor.read(sensorNumber, pinNumber, (err, temperature,
                  humidity) => {
      if (err) {
```

```
        //Ejecutado cuando encuentra un error
        console.error("Error al obtener los datos de temperatura
                      y humedad", err);
        this.temp = "err";
        this.humed = "err";
      }
      else{
        //Modifica las variables que se exportan
        if(temperature)
          this.temp = temperature;
        if(humidity)
          this.humed= humidity;
       }
      //Se finaliza la ejecución de la función añadiendo los
      valores del callback
      callback(temperature, humidity);
     });}
};
```

Figura 23: Función "capture".

4.2.2. Exposición de características BLE para la implementación de la interfaz de gestión del dispositivo BLE

En este apartado se describen las características BLE expuestas por el dispositivo BLE que definen su comportamiento.

Al trabajar con Bleno se ha reutilizado el código facilitado que implementan las clases necesarias para la utilización de servicios y características de los mismos. Además, facilita métodos para la lectura, escritura y subscripción de los servicios.

Por cada funcionalidad que se ha implementado, se ha definido una característica nueva. Dependiendo de su finalidad, la característica puede ser de subscripción, lectura o escritura.

En total se ha creado:

- Un servicio de escritura, que permite cambiarle el nombre al sensor.
- Dos servicios de suscripción que permiten: la emisión de datos al OCB y a la aplicación móvil mediante BLE, y la interrupción de emisión de datos al OCB.

Se ha utilizado el método "startAdvertising" de Bleno para anunciar periódicamente el nombre del dispositivo, además, se envía un indicador que informa sobre el estado de la subscripción del OCB. La aplicación móvil, receptora de la información, muestra el nombre del dispositivo y traduce el indicador en un icono visual. Se ha conseguido de esta forma, conocer el estado del sensor BLE sin necesitar establecer la conexión previamente, posibilitando que esta funcionalidad sea factible cuando el número de sensores aumenta.

En la figura 24 se muestra la definición de una característica que permite el envío de datos a través de BLE.

```
//Emite datos tanto al OCB como al dispositivo movil a través de BLE
var NotifyOnlyCharacteristic = function() {
  NotifyOnlyCharacteristic.super_.call(this, {
    uuid: '1800',
    properties: ['notify','read']
  });
};
```

Figura 24: Definición característica BLE

El "uuid" identifica a la característica y se utiliza en el dispositivo móvil para acceder a él.

Para poder identificar el tipo de muestra (temperatura o humedad) recibida mediante BLE en el dispositivo móvil, se ha antepuesto el valor emitido con una "T"; si el dato corresponde a una muestra de temperatura, o una "H"; si corresponde a una de humedad.

Una vez suscrito a la característica, se ejecuta de forma periódica la función "capture" (detallada en el apartado 4.2.1) que obtiene las muestras de temperatura y humedad y se procede a su envío a través de BLE mediante la función "updateValueCallback" definida en la librería bleno (Figura 25).

```
sensor_data.capture(function (temp, humed) {
  console.log("Datos enviados por Bluetooth: \n");
 //Obtenemos las muestras de temperatura y humedad
 humedad = humed;
 temperatura = temp;
 //Anteponemos una T si la muestra es de temperatura
 data_temp.write('T' + temperatura.toString(),0);
 //Enviamos por BLE
 updateValueCallback(data_temp);
 console.log('Temperatura: ' + temperatura +'\n');
 //Anteponemos una H si la muestra es de humedad
 data_humed.write('H' + humedad.toString(),0);
 //Enviamos por BLE
 updateValueCallback(data_humed);
 console.log('Humedad: ' + humedad +'\n');
 });
```

Figura 25: Anteposición de marca al enviar datos por BLE

La funcionalidad para cambiar el nombre al sensor se define en una característica de tipo escritura que permite el envío del nombre por parte del dispositivo móvil.

Para ello se comprueba la existencia de un archivo temporal (el nombre del archivo se almacena en la constante "configName") almacenado en la raspberry pi que contiene el nombre del dispositivo BLE. Si el fichero existe, se elimina y se vuelve a crear incluyendo el nuevo nombre. Una vez finalizado el proceso de actualización del nombre en el fichero temporal, se utiliza la función "readName" que envía dicha información a broadcast.

En la figura 26 se muestra el código ejecutado cuando nos subscribimos a la característica de escritura e intentamos crear el fichero temporal.

```
WriteOnlyCharacteristic.prototype.onWriteRequest = function(data, off
set, withoutResponse, callback) {
  //Probamos a abrir el archivo que contiene el nombre
```

```
  fs.open(constant.configName, 'wx', (err, fd) => {
    if (err) {
      //Si el fichero existe
      if (err.code === 'EEXIST') {
        //Borramos el archivo temporal
        fs.unlinkSync('./' + constant.configName);
        //Creamos el fichero con el nuevo nombre
        fs.open(constant.configName, 'wx', (err, fd) => {
          fs.appendFile(constant.configName, data.toString(), functio
n (err){
            if(err)
            console.log("Error en el cambio de nombre")
            else{
            bleno.stopAdvertising();
            readName();
            console.log("Cambio de nombre satisfactorio");
            }});
        });
          return;
       }
    }
```

Figura 26: Función para cambio nombre (I)

Si el archivo temporal existe lo elimina y vuelve a crear incluyendo los datos enviados desde el dispositivo móvil.

Si el fichero no existe lo genera y llama a la función "readName" (figura 27).

```
 else{
    //Si el fichero no existe, se crea
    fs.appendFile(constant.configName, data.toString(), function (err
) {
        if (err) {
```

```
          //Error al crear el fichero
        } else {
            //Dejamos de enviar los datos a broadCast y procedemos a
cambiar el nombre del dispositivo
            bleno.stopAdvertising();
            readName();

        }
      });
    }
```

Figura 27: Función para cambio nombre (II)

La función "readName", como se muestra en la figura 28, se limita a leer el nuevo nombre aportado y actualizar el valor enviado a difusión.

```
//Función que lee el nombre del dispositivo BLE en el fichero
function readName(){
  //Intentamos leer el fichero
  fs.readFile('configName.txt', 'utf-8', (err, data) => {
    if(err) {
      console.log('error: ', err);
    } else {
      //Una vez obtenido el nombre del dispositivo, se actualiza los
valores enviados a broadcast
      bleno.startAdvertising(data + " :" + _ocb_state, ['ffffffffffff
ffffffffffffffffff0']);
    }
  });
};
```

Figura 28: Función readName

Además del nombre se envía el estado de subscripción del OCB como se ha indicado con anterioridad.

4.2.3. Envío de datos al OCB

El envío de datos al OCB se realiza mediante la función "ocbService" de forma periódica. Además para el correcto funcionamiento del sistema se realizan dos comprobaciones:

- Primeramente se comprueba la existencia de una subscripción que permita al OCB, informar los cambios en las entidades a Cygnus para que almacene los datos de forma persistente. En caso de no existir la suscripción, se crea.
- Se lee el nombre del dispositivo BLE y se comprueba la existencia de la entidad asociada en el OCB. En caso de no existir se crea una nueva entidad.

Al finalizar las comprobaciones se procede a la actualización de la entidad en el OCB.

Este comportamiento se muestra como diagrama de flujo en la figura 29.

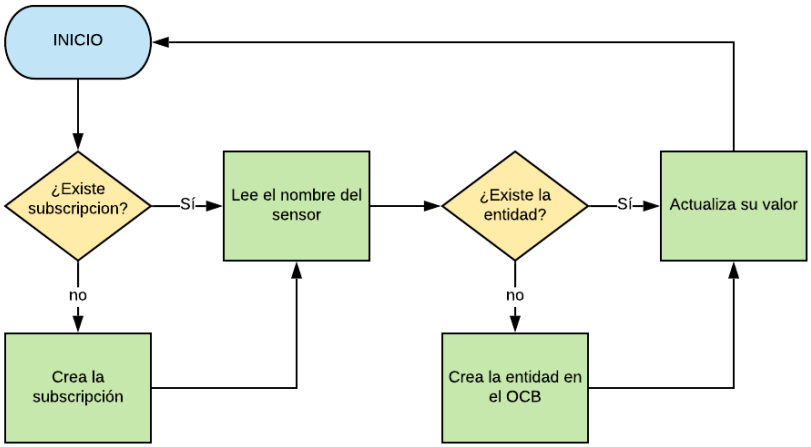

Figura 29: Flujo seguido en la actualización de entidades

En la figura 30 se muestra el código desarrollado.

```
//Emite datos al OCB sgun variable _ocb_state
ocbService = function(){
  console.log("Servicio de envio al OCB funcionando");
```

```
  //Mientras _ocb_state es true, se ejecuta cada 10 minutos
  setInterval(function() {
    if(_ocb_state == true){
      //Utilizamos la función capture del fichero sensor.js
      sensor_data.capture(function (temp, humed) {

      console.log("Datos enviado a BBDD: \n");
      //Obtenemos las muestras de temperatura y humedad
      humedad = humed;
      temperatura = temp;
      console.log('Temperatura: ' + temperatura +'\n');
      console.log('Humedad: ' + humedad +'\n');
      //Comprobamos si existe la suscripción
      getSubscription(function (resp) {
        if(_existe_subs){
          console.log("Subscripción ya creada.");
        }
        else{
          console.log("Subscripción no creada, procedemos a crearla..
.")
          CreateSubscription();
        }
        //Comprobamos el nombre del dispositivo BLE
        lecturaId(function (resp) {
          console.log("Lectura del nombre realizada");
          //Comprobamos si existe una entidad para el dispositivo BLE
          //Segun respuesta, creamos la entidad o la actualizamos
          getEntidad(function (resp) {
            if(this._existe){
              if(_existe_arrancar){
                _existe_arrancar=false;
                CreateEntidad(humedad, temperatura);
              }
              else
```

```
                UpdateEntidad(humedad, temperatura);
              }
            else{
              console.log("Entidad no creada, procedemos a la creacio
n...")
              CreateEntidad(humedad, temperatura);
              _existe_arrancar=false;
            }
          });
        });
      });
    });
    }
  }, 600000);
};
```

Figura 30: Función ocbService

Para realizar las diferentes acciones descritas se utilizan peticiones HTTP REST.

En la figura 31 se detalla el envío de una petición para la actualización de los valores de una entidad:

```
 //Creamos la petición para actualizar la entidad
  let patch_req = http.request(constant.patch_options, function(res)
{
    res.setEncoding('hex');
    res.on('data', function (chunk) {
    });
  });
 //Enviamos la petición
  patch_req.write(JSON.stringify(constant.patch_data));
  patch_req.end();
```

Figura 31: Envío petición REST

La variable “path_options” se define de forma dinámica y se muestra en la figura

32.

```
//Actualizamos los valores de temperatura y humedad
  constant.patch_data.humidity.value = humedad;
  constant.patch_data.temperature.value = temperatura;
  //Configuramos la URI a utilizar de OCB
  constant.patch_options.path =  '/v2/entities/' +
                                 entidadActualizar + '/attrs';
  //Actualizamos la longitud en la cabecera
  constant.headers['Content-length'] = Buffer.byteLength(
    JSON.stringify(constant.patch_data));
  constant.patch_options.headers = constant.headers;
```

Figura 32:Variable “path_options”

En el apartado 7.1.2 se detalla la API NGSI V2 que utiliza OCB para la actualización, creación y eliminación de entidades y suscripciones.

5. APLICACIÓN DESARROLLADA: APP DISPOSITIVO MÓVIL

Se procede a continuación a describir la aplicación móvil híbrida desarrollada que permite interaccionar con el dispositivo BLE a través de la tecnología BLE. Para ello se usa el servicio y características expuestas por el dispositivo BLE.

Como se ha indicado anteriormente, la aplicación móvil se ha desarrollado con el FrameWork Ionic y Apache Cordova.

5.1. IONIC FrameWork

Como se ha indicado con anterioridad Ionic es un framework gratuito y OpenSource utilizado para el desarrollo de aplicaciones híbridas.

En la figura 33 se muestra la arquitectura de IonicV3 (encuadrada en el marco rojo) y de la nueva versión IonicV4.

Al utilizar Ionic3 en nuestro proyecto, hemos tenido que desarrollar la app usando Angular y Apache Cordova.

Tras la salida de IonicV4, podemos utilizar también React, VueJs. Esto es posible gracias a que IonicV4 utiliza WebComponents siendo indiferente la tecnología utilizada.

Figura 33: Arquitectura IONIC Framework

Podemos compilar nuevos WebComponents dentro de Ionic gracias a StencilJs.

En el siguiente punto explicamos la utilidad de Apache Cordova y tras la salida de IonicV4 la alternativa creada por el equipo de Ionic llamada Capacitor.

5.2. Apache Cordova

Una vez desarrollado el código, Ionic es capaz de comunicarse con los elementos nativos del teléfono gracias a Apache Cordova, que a efectos prácticos sirve de puente entre la Web View y los elementos Hardware del dispositivo.

El equipo de Ionic ha lanzado un nuevo Puente llamado Capacitor. Éste tiene la misma finalidad que Apache Cordova, pero diseñado específicamente para Ionic lo que aumenta su rendimiento.

5.3. Interfaz de usuario

Para la implementación de la interfaz de usuario, se ha desarrollado una aplicación móvil compuesta por dos componentes principales:

- Componente Home: Permite la visualización de los dispositivos BLE al alcance del dispositivo móvil. También se ha desarrollado una funcionalidad que permite la visualización de forma gráfica (en forma de indicador de color) del estado de la emisión de datos por parte del dispositivo BLE al OCB.
- Componente Detail: Este componente permite la interacción del operario a través de la aplicación móvil con el dispositivo BLE seleccionado en el componente Home

Toda la información de los dispositivos BLE se almacena en el propio dispositivo BLE permitiendo que dos operarios trabajen sobre los dispositivos de forma simultánea, sin la necesidad de almacenar dicha información en un servidor.

5.3.1. Código

A continuación se procede a explicar cómo se ha desarrollado la funcionalidad de la aplicación móvil a través de los dos componentes principales.

- Componente Home: Muestra la página inicial de la aplicación, encargada de escanear los dispositivos BLE disponibles y mostrar un indicador del estado de la comunicación entre el dispositivo BLE y el OCB.
- Component Details: Además de mostrar los datos de temperatura y humedad que está obteniendo el dispositivo BLE seleccionado en el componente Home, permite modificar el estado de la comunicación entre el dispositivo y el OCB, y gestionar el nombre del dispositivo para poder identificar correctamente las muestras que está obteniendo.

5.3.2. Componente Home

Encargado de escanear los dispositivos BLE disponibles, además de gestionar el indicador enviado por la raspberry pi que muestra el estado de envío de información al OCB.

Para ello comprueba si el dispositivo encontrado es nuevo o se había escaneado con anterioridad, en caso de no ser nuevo lo incluye en la lista para su representación por pantalla. Si el dispositivo encontrado estaba incluído en la lista, actualiza sus valores (nombre del dispositivo, intensidad de la señal e indicador de envío de datos al OCB).

En la figura 34 se muestra la función "scan" ejecutada al pulsar sobre el botón homónimo en la vista de la aplicación.

```
//Método ejecutado al pulsar el botón Scan
scan() {
  console.log('Scanning for Bluetooth LE Devices');
  //Comprueba si existe un escaneo de dispositibos BLE en proceso
  this.ble.isScanning().then(result => {
    //Si no existe un escaneo activo, procede a buscar dispositivos
    if(! result.isScanning) {
      this.ble.startScan({}).subscribe(
      device => this.onDeviceDiscovered(device),
      error => this.scanError(error.message)
          );
        }
      }
    );
  //Añade información por pantalla
  setTimeout(this.setStatus.bind(this), 5000, 'Scan complete');
}
```

Figura 34: Función scan

"ble" es la variable resultante de la importación del módulo de Ionic que incluye los métodos necesarios para operar con la tecnología BLE.

El resultado de "isScanning" es verdadero si sigue escaneando o false si ha terminado el escaneo. Su comprobación es necesaria para la eliminación de errores de ejecución provocados por intentar escanear dispositivos mientras ya había un escaneo en curso.

Una vez se encuentra un dispositivo se utiliza la función "onDeviceDiscovered", mostrada en la figura 35 y encargada de gestionar el dispositivo encontrado, incluyéndolo en la lista anteriormente mencionada o actualizando sus valores.

```
//Método ejecutado cuando al encontrar un dispositivo BLE
  onDeviceDiscovered(device) {
    this.ngZone.run(() => {
      //El dispositivo BLE encontrado es nuevo
      if(String(typeof(device.name)) == 'string' && device.name != 'U
nnamed' && device.name != '' && device.name != ' '
      ){
        /* Inicializa la variable para representar el dispositibo BLE
           y su estado de envío al OCB por pantalla */
        let deviceprinted = this.DevicePrinted = {deviceStatus :devic
e, state: null};
        //Comprueba si el dispositivo BLE encontrado ya se había
        escaneado a través del servicio Connection
        if(this.connectionService.getAddresses().indexOf(device.addre
ss) != -1)
        {
          //El dispositivo BLE encontrado ya se había escaneado
          //Comprueba el estado de envío al OCB del dispositivo
          encontrado
          if(device.name.split(' :true').length == 2){
            device.name = device.name.replace(' :true', '');
            deviceprinted.state = true;

          }
          //Comprueba el estado de envío al OCB del dispositivo
          encontrado
          if(device.name.split(' :false').length == 2){
            device.name = device.name.replace(' :false', '');
            deviceprinted.state = false;
          }
          //Actualiza el estado del dispositivo encontrado
          this.devices[this.connectionService.getAddresses().indexOf(
device.address)] = deviceprinted;
```

```
        }
        //El dispositivo BLE encontrado no se había escaneado
        else{
          if(deviceprinted.deviceStatus.name.split(' :true').length =
= 2){
            deviceprinted.deviceStatus.name = deviceprinted.deviceSta
tus.name.replace(' :true', '');
            deviceprinted.state = true;
          }
          if(deviceprinted.deviceStatus.name.split(' :false').length
== 2){
            deviceprinted.deviceStatus.name = deviceprinted.deviceSta
tus.name.replace(' :false', '');
            deviceprinted.state = false;
          }
          /* Para almacenar el dispositivo BLE, tiene que estar
             enviando el estado de envío de datos al
             junto con el nombre */
          if( deviceprinted.state == false || deviceprinted.state ==
true){
            //Almacena el dispositivo BLE en la variable que utiliza
            la vista para mostrarlos
            this.devices.push(deviceprinted);
            /* Utiliza el servicio para almacenar el dispositivo BLE
               y si se encuentra nuevamente se actualiza su estado */
            this.connectionService.setAddresses(device.address);
          }
        }
       }
      });
  }
```

Figura 35: Función ejecutada al escanear un dispositivo

En la figura 36, se muestra la vista relacionada al código anterior:

```
<ion-content padding class="bg-image">
  <ion-list class="fg-list">
    <button ion-
tem *ngFor="let device of devices" (click)="deviceSelected(device.dev
iceStatus)">
      <ion-row>
        <ion-item col-10>
          <h2>Sensor: {{ device.deviceStatus.name || 'Unnamed' }}</h2
>
          <p>{{ device.deviceStatus.address }}</p>
          <p>RSSI: {{ device.deviceStatus.rssi }}</p>
        </ion-item>
        <ion-item col-
2 *ngIf="device.state == false" class="stateMark">
          <ion-icon name="close-circle" color="danger"></ion-icon>
        </ion-item>
        <ion-item col-2 *ngIf="device.state == true">
          <ion-icon name="checkmark-
circle" color="primary" class="stateMark"></ion-icon>
        </ion-item>
      </ion-row>
    </button>
  </ion-list>
</ion-content>
```

Figura 36: Función para emitir broadcast

Se consulta el valor de la variable “state” del objeto “device” con el fin de representar por pantalla un icono azul en caso de estar enviando datos al OCB o en rojo en caso contrario.

En la figura 37 podemos ver la vista del componente descrito.

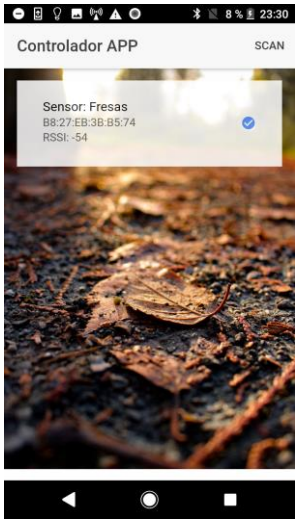

Figura 37: Vista componente Home

5.3.3. Componente Detail

Este componente es el encargado de realizar todas las acciones disponibles en el sensor seleccionado en el componente Home.

Nada más acceder a dicho componente, se ejecuta la función "connect" procediendo a establecer la conexión BLE (previamente a establecer dicha conexión, se tiene que comprobar que el estado del adaptador BLE del dispositivo móvil está listo para su uso).

En caso de conectarse al dispositivo BLE se proporcionan los métodos necesarios para proceder al envío de datos o el cambio de nombre del sensor.

En la figura 38 se muestra el código que comprueba el estado del adaptador BLE y procede a la conexión con el dispositivo.

```
this.ble.initialize(this.device).then(
    () => {
      //Procede a la conexión con el dispositivo BLE
      this.ble.connect(this.device).subscribe(
        //Si encuentra un dispositivo ejecuta el método onConnected
```

```
        peripheral => this.onConnected(peripheral),
        error => {
          //Si existe un error con la conexión, cierra el intento y
          vuelve a la pantalla anterior
          this.ble.close(this.device).then((deviceInfo) => {
            this.navCtrl.pop();
            })
          .catch((error) =>
            console.log("Error closing connection" + error.message
         );
        }
    );
  }
);
```

Figura 38: Función connect

Para cada característica expuesta por el sensor existe una función que la gestiona a través de BLE:

- Cambio de nombre del dispositivo: Recibe por pantalla el nuevo nombre del dispositivo y procede a utilizar la característica de escritura del dispositivo BLE que ejecuta el cambio (el código desarrollado se muestra en la figura 39).

```
cambiaNombre(){
  //Comprueba que no sea nulo
  if(this.nameInput){
    //Añade el nombre introducido al Buffer
    this.nameBuf = new Buffer(this.nameInput);
    //Configura la variable para la modificación
    de la caracteristica del servicio
    this.paramWriteCharacteristic = { value: this.nameBuf.toString
    ('base64'),
    type: "string",
```

```
    characteristic: WRITE_CHARACTERISTIC,
    service: RPI_SERVICE, address: this.device.address };
    //Ejecuta la modificación de la característica
    this.ble.write(this.paramWriteCharacteristic)
    .then(() =>
      console.log("Write funcionando")
    )
    .catch(e =>
      console.log("Write no funcionando", e.message)
    );
  }
}
```

Figura 39: Función para cambiar nombre

Dónde "RPI_SERVICE" y "WRITE_CHARACTERISTIC" son identificadores únicos.

- Recepción de datos de temperatura y humedad: Para recibir los datos de temperatura y humedad se procede a subscribirse al servicio creado en el dispositivo BLE del tipo notificación. Una vez recibida la información se procede a decodificarla de Base64. Estos datos se muestran por pantalla (el código desarrollado se muestra en la figura 40).

```
startDataSubscription(){
  this.setStatus("Enviando por BLE y al OCB");
  //Inicializa la variable para la conexión al
  servicio BLE
  this.paramSensorService = {
    characteristic : SENSOR_CHARACTERISTIC,
    service : RPI_SERVICE,
    address : this.device.address
  };
  //Actualiza variables de estado de conexión
  del dispositivo BLE
```

```
this.dataSubscribed = true;
this.dataBLESubscribed = true;
this.connectionService.pushState(this.device.address);
//Procede a la conexión con el servicio BLE
this.ble.subscribe(this.paramSensorService).subscribe(
  objectSuscribed => {
    if(String(typeof(JSON.parse(JSON.stringify
      (objectSuscribed)).value)) != 'undefined'){
        //El sensor envia los datos de Temperatura anteponiendo
        una T, los de humedad anteponiendo una H.
        //Identificamos el tipo de dato y eliminamos
        la marca
        //Los datos recibidos están codificados en
        base 64
        this.data = JSON.parse(JSON.stringify
                              (objectSuscribed)).value;

      if(this.b64DecodeUnicode(this.data).slice(0,1)
        == 'T'){
        this.data_temp =  this.b64DecodeUnicode
                          (this.data).slice(1);
      }
      if(this.b64DecodeUnicode(this.data).slice(0,1)
      == 'H'){
        this.data_humed = this.b64DecodeUnicode
                          (this.data).slice(1);
      }
    }
    console.log('Encodedbytes to String: ' + this.data);
  },
  err =>{
    console.log('Suscribe Error: ' + err.message)
  }
);
```

```
}
```

Figura 40: Función para emitir los datos de temperatura y humedad

La constante “sensor_characteristic” tiene como valor otro identificador único.

Para dejar de enviar datos por BLE cancelamos la subscripción del servicio anterior de forma similar a como se crea dicha subscripción (el código se muestra en la figura 41):

```
stopDataSubscription() {
  this.setStatus("Sin enviar al OCB");
  //Configuramos la variable para utilizar la característica
  del servicio
  this.paramCygnusSubscr = {
    characteristic: CYGNUS_CHARACTERISTIC,
    service: RPI_SERVICE,
    address: this.device.address
  };
  //Procedemos a conectarnos a la característica para
  que la variable _ocb_state del dispositivo BLE se reevalua a false
  this.ble.subscribe(this.paramCygnusSubscr).subscribe(
    objectSuscribed => {
      //Completamos la modificación desuscribiéndonos del servicio
      this.ble.unsubscribe(this.paramCygnusSubscr)
      .then(() =>
        console.log("Se finaliza la eliminación de subscripción")
       ).catch(err =>
         console.log("Error al parar el envio al OCB" + err.message)
         );
         //Actualiza variables de estado de conexión del
         dispositivo BLE
         this.dataSubscribed = false;
         this.connectionService.pullState(this.device.address);
       },
```

```
    err =>{
      console.log('Error al eliminar las subscripciones a
                  entidades: ' + err.message)
    }
  );
}
```

Figura 41: Función para cancelar el envío BLE

- Para parar el envío de datos al OCB se utiliza la función mostrada en la figura 42 "stopDataSubscription()", que cancela la suscripción de Cygnus al OCB para el sensor seleccionado (no se almacenan los datos en la BBDD) y provoca además, que el sensor no envía datos al OCB:

```
stopDataSubscription() {
  this.setStatus("Sin enviar al OCB");
  //Configuramos la variable para utilizar la característica
  del servicio
  this.paramCygnusSubscr = {
    characteristic: CYGNUS_CHARACTERISTIC,
    service: RPI_SERVICE,
    address: this.device.address
  };
  //Procedemos a conectarnos a la característica para
  que la variable _ocb_state del dispositivo BLE se reevalua a false
  this.ble.subscribe(this.paramCygnusSubscr).subscribe(
    objectSuscribed => {
      //Completamos la modificación desuscribiéndonos del servicio
      this.ble.unsubscribe(this.paramCygnusSubscr)
      .then(() =>
        console.log("Se finaliza la eliminación de subscripción"),
      ).catch(err =>
        console.log("Error al parar el envio al OCB" + err.message)
      );
      //Actualiza variables de estado de conexión del dispositivo BLE
        this.dataSubscribed = false;
```

```
        this.connectionService.pullState(this.device.address);
    },
    err =>{
      console.log('Error al eliminar las subscripciones
                  a entidades: ' + err.message)
    }
  );
}
```

Figura 42: Función para cancelar el envío al OCB

La constante “cygnus_characteristic” tiene como valor el identificador único que se ha asignado en el dispositivo BLE a la característica que modifica el comportamiento del dispositivo BLE con el OCB.

En la figura 43 se observa la visualización del componente descrito.

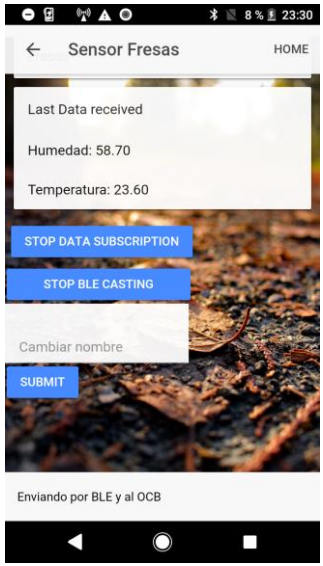

Figura 43: Vista componente Detail

6. APLICACIÓN DESARROLLADA: PÁGINA WEB

Se ha desarrollado una página web utilizando Angular, que se comporta como sumidero de toda la información obtenida por los diferentes dispositivos BLE.

Se ha desarrollado una primera vista, que permite la visualización de los dispositivos BLE existentes. Por cada uno, se implementa un botón que permite eliminarlo, otro que permite visualizar los máximos y mínimos de temperatura y humedad y un último botón que permite navegar a la segunda vista.

La segunda vista desarrollada muestra por pantalla todos los datos obtenidos por el dispositivo BLE y permite el filtrado de dicha información en intervalos de fechas seleccionables por el usuario.

En la figura 44 se representa la página desarrollada.

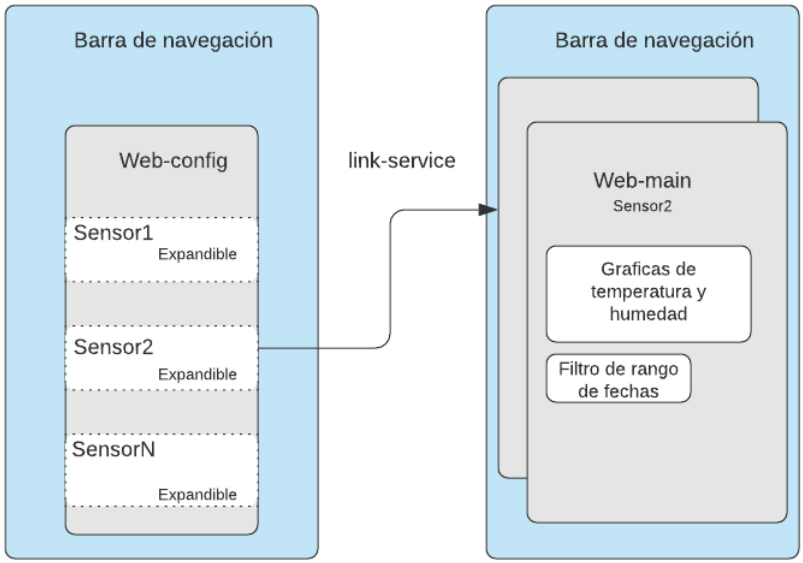

Figura 44: Esquema página web desarrollada

6.1. Template: Barra de navegación

Componente principal de la página, permite navegar entre las dos vistas desarrolladas. Para ello, se utiliza la directiva "router-outlet" que define un espacio donde se inyectan las diferentes vistas de la página web.

Hemos definido por tanto, un encabezamiento principal, que no varía durante la navegación, y una zona donde se cargan los diferentes componentes a los que se quieren acceder.

En la figura 45, se muestra la definición de una sección en el documento HTML principal dónde se utiliza la directiva "router-outlet" para inyectar los diferentes componentes.

```
<div class="container">
  <router-outlet></router-outlet>
</div>
```

Figura 45: Utilización etiqueta "router-outlet"

6.2. Vistas página web

Como se ha indicado al principio del apartado, se han desarrollado dos vistas: la primera de ellas que visualiza los sensores disponibles, y una segunda que permite visualizar toda la información disponible del sensor seleccionado.

Cada vista se inyecta gracias a la directiva "router-outlet" dentro del componente "nav-bar" representado en la Figura 44 como barra de navegación. Proporciona fluidez en la navegación.

A continuación, explicamos al detalle las vistas utilizadas (el funcionamiento de los servicios descritos a continuación se explican en el apartado 7.3 Servicios).

6.2.1. Vista: Web-config

La vista inicial muestra los diferentes dispositivos BLE con datos almacenados en la BBDD a través del servicio sensor.service.

Implementa dos botones por cada sensor existente, uno que procede al borrado del sensor y de toda la información que contiene, y otro que permite la visualización de los máximos y mínimas en las muestras de temperatura y humedad obtenidas.

La obtención del listado de dispositivos BLE se ejecuta en un primer momento a través de la función definida en Angular "ngOnInit" que se ejecuta una vez la vista ha sido cargada con éxito (se muestra en la figura 46).

```
//Ejecutada al cargar el componente
ngOnInit() {
  //Obtiene la lista de dispositibos BLE
  this.sensorService.getSensors().subscribe(
    res => {
      //Variable que los muestra por pantalla a través de la vista
      this.sensorsList = JSON.parse(JSON.stringify(res)).SensorList;
    },
    err => console.error(err)
  );
}
```

Figura 46: Función ngOnInit

En la figura 47 observamos la visualización de la vista.

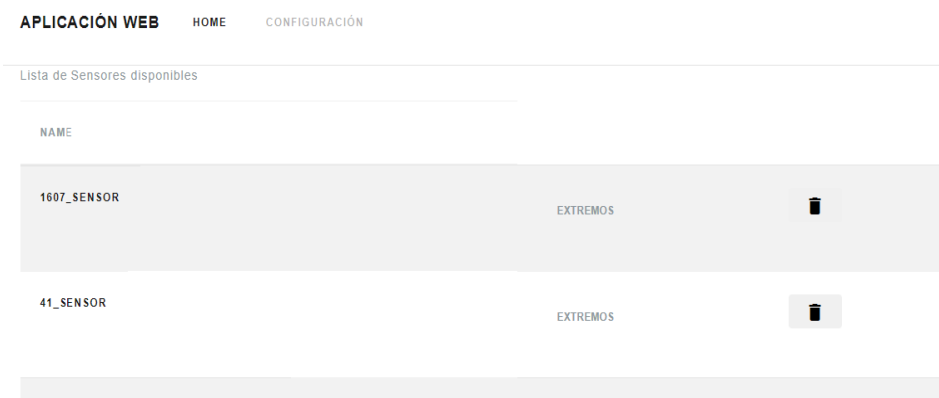

Figura 47: Vista componente Web-config

6.2.1.1. Expandible

Componente hijo de "Web-config" utilizado para mostrar los máximos y mínimos

absolutos de un sensor dado al pulsar el botón “Extremos”. Si volvemos a pulsar dicho botón la información se oculta.

Hace uso del servicio “entity.service” detallado en el apartado 7.3.2. entity.service para obtener los datos del sensor seleccionado y para la obtención de máximos y mínimos absolutos de dichos datos obtenidos.

En la figura 48 se muestra una captura de pantalla de la aplicación donde se comprueba el funcionamiento del componente.

Figura 48: Vista componente Expandible

6.2.2. Vista: Web-main

Vista principal que muestra los datos asociados a un sensor específico. Contiene dos gráficas donde se representan dichos datos. Podemos filtrarlos por un rango de fechas en concreto.

Utiliza el servicio entity.service para recuperar la información alojada en BBDD del dispositivo seleccionado (recuperado a través del servicio anteriormente mencionado: link.service) en la vista Web-config.

Se limita a recuperar dichos datos y gestionarlos para mostrarlos por pantalla. Ofrece dos campos donde podremos indicar el rango de fechas a consultar. Para su utilización nos servimos de los dos botones existentes.

Uno de ellos muestra todos los datos recogidos por el sensor y almacenados en la BBDD, el otro botón filtra la información obtenida según el rango de fechas anteriormente seleccionado.

En la figura 49 se muestra el código desarrollado para mostrar por pantalla los datos

de Temperatura, así como el código relacionado con la selección de fecha de inicio a filtrar. El código relacionado con los datos de Humedad, así como la selección de la fecha fin a filtrar es similar.

El componente utiliza dos funciones principales:

- ngOnInit(): Ejecutada en un primer momento tras cargar exitosamente el componente. Procede a utilizar el servicio REST expuesto por el "Back End" para consultar los datos almacenados del dispositivo BLE seleccionado. Además, ejecuta una función auxiliar definida en el servicio "entity.service" que modifica los datos obtenidos para poder representarlos fácilmente por pantalla. La función se muestra en la figura 50.
- buscarIntervaloFecha(): Filtra los valores obtenidos según los valores "dataRangeBegin" y "dateRangeEnd" indicadas en la vista (Figura 51).

```
<div style="display: block" class="column">
  <p>Temperatura</p>
  <canvas baseChart
    [datasets]="barChartDataTemperature"
    [labels] = "barChartLabels"
    [options]="barChartOptions"
    [legend]="barChartLegend"
    [chartType]="barChartType">
  </canvas>
</div>
<p>
  <mat-form-field>
     Fecha inicio
    <input matInput [matDatepicker]="dataRangeBegin" [(ngModel)]="selec
tedDataRageBegin">
    <mat-datepicker-toggle matSuffix [for]="dataRangeBegin">
    </mat-datepicker-toggle>
    <mat-datepicker #dataRangeBegin>
    </mat-datepicker>
  </mat-form-field>
```

Figura 49: Código para mostrar la temperatura

```
//Ejecutada al cargar el componente
ngOnInit() {
  //Obtiene los datos relacionados con un sensor.
  this.entityService.getEntity(this.sensorSelected).subscribe(
    res => {
      //Utiliza el servicio entityService
      this.entity = this.entityService.parseaDatos(JSON.parse(JSON.stri
ngify(res)).EntitiesList);
    },
    //Devolvemos el error obtenido
    err => console.error(err)
  );
}
```

Figura 50: Función ngOnInit web main

```
buscaIntervaloFecha(){
  let dataTemperatureDateAux = this.entity.dataTemperatureDate

  /*Obtenemos las fechas de las muestra de temperatura obtenidas
  incluidas en el rango de fechas seleccionado en la vista*/
  var resultBegin = dataTemperatureDateAux.filter(dateFiltered => (Date
.parse(this.convertDate(dateFiltered)) > this.selectedDataRageBegin.get
Time()) &&
                                                                  Date
.parse(this.convertDate(dateFiltered)) <  this.selectedDataRageEnd.getT
ime());
  let longitud = resultBegin.length;
  /*Obtenemos los indices que marcan el intervalo seleccionado
  en el conjunto de fechas de las muestras obtenidas*/
  let firstIndex = this.entity.dataTemperatureDate.indexOf(resultBegin[
0]);
  let lastIndex = this.entity.dataTemperatureDate.indexOf(resultBegin[l
ongitud - 1]);
```

```
  this.dataTemperatureAux = [];
  this.dataHumidityAux = [];

  /*Obtenemos los datos de temperatura y humedad con fecha
  de obtención incluidas en los indices calculados*/
  for ( var i = firstIndex; i <= lastIndex; i++ ){
    this.dataTemperatureAux.push(this.entity.dataTemperature[i]);
    this.dataHumidityAux.push(this.entity.dataHumidity[i]);
  }
  //Almacenamos las fechas obtenidas para su representación en el eje X
 de las gráficas
  this.barChartLabels = resultBegin;
}
```

Figura 51: Función buscarIntervaloFecha

En la figura 52 observamos la vista del componente filtrando por un intervalo de fechas.

Figura 52: Vista componente Web-main

6.3. Servicios

Los servicios son componentes adicionales en un proyecto angular que relacionan los diferentes componentes principales, proporcionando una interfaz entre ellos que permite realizar diversas acciones en los componentes donde hayan sido inyectados.

Tenemos tres Servicios:

- sensor.service: Proporciona diferentes métodos para trabajar con los sensores registrados en el sistema.
- entity.service: Proporciona diferentes métodos para trabajar con los datos recogidos por cada sensor individualmente.
- link.service: Proporciona una conexión entre el componente web-config y web-main almacenando el sensor seleccionado y recuperándolo respectivamente.

6.3.1. Sensor.service

Aporta dos funciones que soportan la conexión con el Back End para recuperar el listado de sensores disponibles, así como para la eliminación de un sensor en concreto del sistema. Esto se realiza a través del módulo HttpClient utilizando las rutas disponibles por el Back End.

El código del servicio desarrollado se muestra en la figura 53.

```
//Se utiliza el módulo HttpCliente para realizar las peticiones HTTP
constructor(private http: HttpClient) {
 }
 //Obtiene el listado de dispositivos BLE disponibles
 getSensors(): Observable<Object>{
  return this.http.get(this.API_URI +'/sensor');
}
 //Elimina el sensor seleccionado
 deleteSensor(sensor): Observable<Object>{
  return this.http.delete(this.API_URI + '/sensor/' + sensor);}
```

Figura 53: Módulo HttpClient

6.3.2. Entity.service

Este servicio aporta una función que permite obtener los datos obtenidos por el dispositivo BLE (parámetro de entrada de la función). Además, crea dos métodos: un primer método que procesa los datos para representarlo por pantallas de forma sencilla, y otro, que permite el cálculo de los máximos y mínimos absolutos para la humedad y temperatura.

El primer método,"parseaDatos()" (mostrada en la figura 54), almacena en el tipo de objetos "entity" los valores recibidos de BBDD:

```
   /*Comprobamos si el dato a leer es de Temperatura o humedad,
     lo almacenamos en el array correspondiente y añadimos la marca de
tiempo recibida*/
    for (var exkey in exjson){
    //Comprobamos si el valor corresponde a una muestra de temperatura
      if(exjson[exkey].attrName == 'temperature'){
        //Almacenamos el valor de temperatura
        entity.dataTemperature.push(exjson[exkey].attrValue);
        //Almacenamos la fecha en la que se obtuvo la muestra
        entity.dataTemperatureDate.push(this.datePipe.transform(exjson[
exkey].recvTime,"dd-MM-yyyy HH:mm:ss"));
      }
      //Comprobamos si el valor corresponde a una muestra de humedad
      if(exjson[exkey].attrName == 'humidity'){
        //Almacenamos el valor de humedad
        entity.dataHumidity.push(exjson[exkey].attrValue);
        //Almacenamos la fecha en la que se obtuvo la muestra
        entity.dataHumidityDate.push(this.datePipe.transform(exjson[exk
ey].recvTime,"dd-MM-yyyy HH:mm:ss"));
      }
  }
```

Figura 54: Función parseaDatos

Implementa un segundo método utilizado por la vista Web-config para recuperar los datos de temperatura y humedad máximos y mínimos absolutos.

En la figura 54 se muestra el código desarrollado para la obtención de máximos absolutos de temperatura para un sensor dado (el código utilizado para obtener los máximos y mínimos absolutos de humedad es similar y se ha omitido):

```
public getEntityMaxMin(entityData: Entity) {
/*Almacenamos los valores de temperatura y humedad como tipo Float para
 posteriormente recuperar
     el máximo y mínimo*/
  var temperature_mapped = Object.keys(entityData.dataTemperature).map(
function(key) {
     return parseFloat(entityData.dataTemperature[key]);
    });

// Recuperamos el máximo y mínimo de temperatura
  maxMin.dataTemperatureMax = Math.max.apply(null, temperature_mapped);
  maxMin.dataTemperatureMin = Math.min.apply(null, temperature_mapped);

// Obtenemos la fecha en el que se produjo el máximo y el mínimo
  if(temperature_mapped.indexOf(maxMin.dataTemperatureMax) >= 0)
    maxMin.dataTemperatureDateMax = entityData.dataTemperatureDate[temp
erature_mapped.indexOf(maxMin.dataTemperatureMax)];
  if(temperature_mapped.indexOf(maxMin.dataTemperatureMin) >= 0)
    maxMin.dataTemperatureDateMin = entityData.dataTemperatureDate[temp
erature_mapped.indexOf(maxMin.dataTemperatureMin)];
    return maxMin
};
```

Figura 55: Función para obtener max y min

6.3.3. Link

El servicio link.service es utilizado por la vista web-main para recuperar el sensor escogido en la pantalla inicial.Modelos

6.4. Modelos

6.4.1. Entity

Se ha creado la interfaz "Entity" para el correcto procesado de la información a través de la función "getEntity()" del servicio "entity.service" (apartado 7.3.2).

Utilizamos los atributos: "attrName", almacena el tipo de la muestra obtenida (temperatura o humedad); "attrValue", en este caso almacena el valor de dicha muestra; "recvTime", almacena la fecha cuando se tomó la muestra.

Su definición se muestra en la figura 55.

```
export interface Entity {
  recvTime?: Date;
  attrName?: string;
  attrValue?: number;
}
```

Figura 56: Interfaz Entity

6.4.2. MaxMin

La interfaz MaxMin se utiliza para almacenar de forma más sencilla los máximos y mínimos aboslutos de los datos obtenidos por un dispostivo BLE. Almacena además la fecha de la muestra. Su definición se muestra en la figura 56.

```
export interface MaxMin {
  dataTemperatureMax: any,
  dataTemperatureDateMax: any,
  dataTemperatureMin: any,
  dataTemperatureDateMin: any,
  dataHumidityMax: any,
  dataHumidityDateMax: any,
  dataHumidityMin: any,
  dataHumidityDateMin: any}
```

Figura 57: Interfaz Maxmin

7. APLICACIÓN DESARROLLADA: APIS UTILIZADAS Y SERVICIO WEB

Este apartado detalla las diferentes APIs utilizadas en el proyecto para poder comunicar los diferentes componentes de la aplicación desarrollada.

Se divide en dos apartados, uno específico para el servicio web y la API que expone, y otro para explicar la API NGSI V2 que expone OCB y que utiliza el dispositivo BLE.

En la figura 58 se muestra las APIs que utiliza los diferentes componentes y como se comunican entre sí.

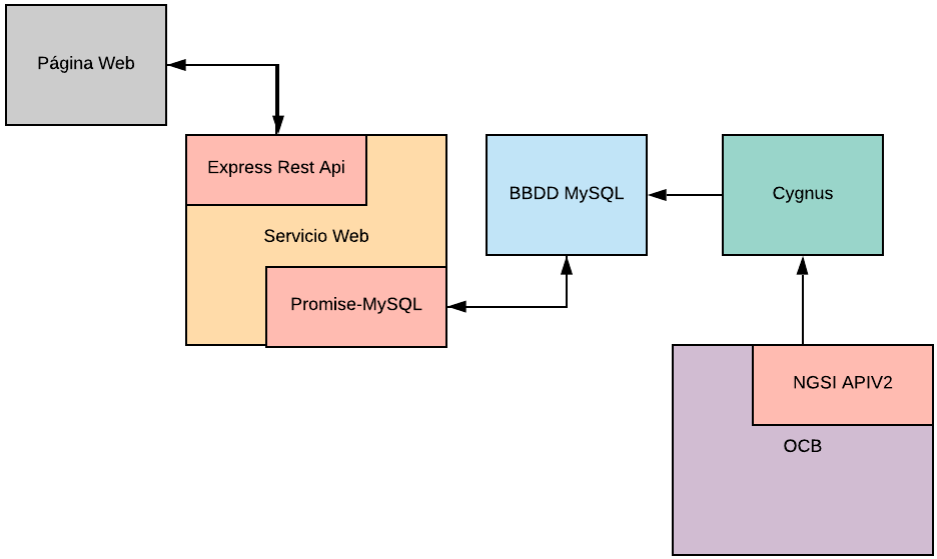

Figura 58: Comunicación entre componentes

7.1. Servicio Web

Se ha desarrollado un servicio web, que permite la visualización de los datos

obtenidos por los dispositivos BLE en un navegador web.

Para ello, el servicio expone una API REST a la página web y componen las siguientes funcionalidades:

- Obtención de los dispositivos BLE.
- Eliminación del dispositivos BLE del sistema y de todos los datos de temperatura y humedad obtenidos por dicho dispositivo BLE.
- Obtención de los datos de temperatura y humedad obtenidos por los dispositivos BLE.

Este servicio WEB se conecta a la BBDD relacional MySQL donde el componente Cygnus del OCB almacena los datos de temperatura y humedad.

En este apartado se procede a explicar con detenimiento el servicio, además, se procede a explicar el modo de funcionamiento de las distintas tecnologías involucradas en el proceso de almacenamiento de la información, y su posterior obtención por la página web.

También procedemos a explicar Express (módulo de Node utilizado para la creación de la API REST) y Promise-MySQL (utilizado para la comunicación con la BBDD).

7.1.1. Definición de API para obtener los datos de BBDD

Se explica a continuación la API expuesta por el back end de la aplicación web para poder comunicar el front end con la BBDD, permitiendo actuar sobre la BBDD desde la página web.

Para ello se divide en dos tablas que expone los diferentes métodos HTTP y la URI a la que se atiende según trabaje con la lista de sensores o con los datos obtenidos de cada sensor.

En la tabla 1 se observa la API expuesta por el back end de la aplicación web desarrollada para consultar los sensores disponibles y la eliminación de los datos almacenados de uno de ellos.

Método	URI	Descripción
GET	/api/sensor	Obtiene el listado de sensores disponibles.

DELETE	/api/sensor/idSensor	Eliminar el sensor cuyo identificador se ha especificado en "idSensor".

Tabla 1: API para la obtención y eliminación de los sensores

En la tabla 2 se muestra la API expuesta para consultar la información de temperatura y humedad obtenida por un sensor en particular.

Método	URI	Descripción
GET	/api/entity/idSensor	Obtiene todos los datos almacenados del sensor cuya id se ha especificado en "idSensor"
DELETE	/api/entity/idSensor/idAtributo	Obtiene los datos de temperatura o humedad (especificados según "idAtributo"; siendo humedad o temperatura) relacionados con el sensor cuya id se ha especificado en "idSensor".

Tabla 2: API para la obtención de los datos de un sensor en particular

El "idSensor" se compone por el nombre que se le ha asignado con el dispositivo móvil y el sufijo "_sensor". Por ejemplo, "parcela8_sensor". Estructura BBDD

La estructura de base de datos básica MySQL que trabaja Cygnus está compuesta por:

- Una tabla por cada entidad diferente existente en el OCB. El nombre de la tabla está compuesto por el nombre del sensor que precede al tipo de sensor. Por ejemplo: "Invernadero1_Sensor".
- En cada tabla existe un registro que contiene información relevante del contexto.

A continuación, se aporta la estructura de la tabla enumerando sus columnas:

- recvTimeTs: Tiempo en formato UNIX en milisegundos.
- recvTime: Tiempo en formato estándar: “aaaa-mm-dd HH:mm:ss”
- entityId: Nombre de la entidad de contexto
- entityType: Tipo de entidad de contexto
- attrName: Nombre de la variable de contexto.
- attrType: Tipo de variable de contexto.
- attrValue: Valor de la variable de contexto.

7.1.2. Módulos Servicio Web

Para la creación de la API REST se ha utilizado el módulo Express, y para manejar la conexión con BBDD se ha utilizado Promise-MySQL.

Ambos son muy populares en el mercado y existe mucha información sobre ellos, siendo sencillo encontrar solución a problemas obtenidos durante el desarrollo. Se encuentran incorporados en el gestor de paquete de node por lo tanto su instalación se realiza de forma sencilla mediante dicho gestor. La instalación de ambos módulos se indica en el Anexo C.

Explicamos a continuación ambos módulos.

7.1.2.1. Express

Se define como “una infraestructura de aplicaciones web Node.js mínima y flexible “ que proporciona la herramienta necesaria para la creación de APIs[13].

A través de los métodos incluídos en express definimos el puerto de escucha y las diferentes rutas para los diferentes recursos servidos. La definición del servidor se muestra en la figura 59.

Se define cada uno de los recursos expuestos por el servidor y los enlaza con el código desarrollado para cada recurso.

```
//Definimos la variable que utiliza el módulo express
this.app = express();
//Definimos el puerto que escucha
this.app.set('port', process.env.PORT || 3000);
```

```
routes(): void {
 //Definimos las rutas a los diferentes componentes
 this.app.use('/', indexRoutes);
 this.app.use('/api/entity', entityRoutes);
 this.app.use('/api/sensor', sensorRoutes);
}
//Método que arranca el servidor en el puerto definido
start() {
  this.app.listen(this.app.get('port'), () => {
    console.log('Server on port:', this.app.get('port'));
  });
}
```

Figura 59: Definición del servidor

Cada recurso en el servidor es una clase incluída en un fichero TypeScript. Se definen dos recursos:

- "entityRoutes": Encargado de enlazar las consultas de BBDD con la URI accedida. Proporcio una puerta de acceso para la información de temperatura y humedad de cada entidad.
- "sensorRoutes": Enlaza las consultas de BBDD con la URI que identifica al recurso. Proporciona una puerta de acceso a las diferentes entidades existentes.

Mostramos en la figura 60 el código que relaciona las consultas a BBDD con las URIS disponibles relacionadas con la obtención de datos de los dispositivos BLE (fichero "entityRoutes").

```
config(): void {
  /*Enlazamos la URI utilizada con el método
  que interactúa con la BBDD */
  this.router.get('/',entityController.list);
  this.router.get('/:sensor_id',entityController.selectOne);
  this.router.get('/:sensor_id/:attribute',entityController.selectOneAttr);
```

Figura 60: Mapeo información de los sensores.

7.1.2.2. Promise-MySQL

Para la conexión con la BBDD MySQL donde se almacenan los datos persistentemente se utiliza la librería "Promise-MySQL" que engloba el módulo MySQL proporcionando asincronismo a las consultas realizadas.

Este asincronismo es fundamental para la correcta ejecución de la aplicación web desarrollada, debido a que permite la gestión de varios procesos de consulta simultáneos contra la BBDD.

A continuación, enumeramos las diferentes consultas que se ejecutan en el servicio Web y mostramos el código desarrollado para implementar la funcionalidad.

7.1.2.3. Conexión con BBDD

Las consultas a BBDD son las siguientes:

- SHOW TABLES: muestra todas las tablas almacenadas (cada tabla almacena los datos de un único sensor)
- DROP TABLE idSensor: borra la tabla con la información de "idSensor".
- SELECT * FROM idSensor: obtiene todos los datos relacionados con el sensor "idSensor"
- SELECT * FROM idSensor WHERE attrName = idAtributo: obtiene los datos de temperatura o humedad (según "idAtributo") relacionados con el sensor "idSensor".

Para la creación de la conexión a BBDD y el posterior tratamiento de la información que contiene se ha creado un módulo exportable que permite su uso desde cualquier otro punto del programa que importe dicho módulo (En la figura 61, mostramos el código utilizado):

```
/*Obtenemos la cadena de conexión definida en
 el fichero ./keys*/
const pool = mysql.createPool(keys.database);
//Nos conectamos a la BBDD
pool.getConnection()
    .then (connection => {
        pool.releaseConnection(connection);
```

```
        console.log('DB is connected');
    });
//Exportamos la conexión como variable
export default pool;
```

Figura 61: Creación de conexión a BBDD

El objeto "pool" almacena la conexión a la BBDD definida en el documento "keys.js" y proporciona los métodos necesarios para realizar las consultas.

Una vez obtenido dicho objeto podemos proceder a ejecutar consultas sql a través del método "query". Se definen dos métodos diferentes, uno que permite listar todos los sensores creados, y otro que permite el borrado de uno en particular. En la figura 62 se muestra el código desarrollado.

```
//Método que obtiene todos los dipositivos BLE disponibles
public async list (req: Request, res: Response): Promise <any> {
  //Ejecutamos la query asíncronamente
  const sensors = await pool.query('SHOW TABLES');
  res.json({"SensorList":sensors});
}
/* Método que elimina el dispositivo BLE seleccionado y
toda las muestras de temperatura y humedad obtenidas */
public async delete (req: Request, res: Response){
  try{
    const sensor = await pool.query('DROP TABLE ' + req.params.sensor_id
);
    res.json ({text: 'Sensor deleted: ' + sensor})
  } catch (e) {
    //Devuelve error si el dispositivo no existe
    res.status(404).json ({text: 'Error deleting: ' + req.params.sensor_
id})
  }
};
```

Figura 62: Controlador a BBDD para la información relacionada con los sensores

Se define además otro método que permite la obtención de los datos de entidad para un sensor en concreto en el fichero “entityController”

Debido a la similitud existente con el código de “sensorController” se ha omitido el código encargado de obtener los datos de temperatura y humedad de los sensores.

7.2. NGSI APIV2

Se explica a continuación la API NGSI V2 que utiliza OCB para la el manejo de las entidades de contexto y las suscripciones a las modificaciones en dichas entidades.

7.2.1. Entidades

Una entidad es una estructuración de datos caracterizada por un conjunto de atributos que almacenan la información procedente de los sensores. Están identificadas por un id unívoco.

En nuestro caso las entidades de tipo Sensor tienen los datos procedentes del sensor y está caracterizado por:

- Id: valor único que representa el nombre del sensor y nos sirve para procesar la información de un sensor específico.
- Type: tipo de entidad.
- Humidity: almacena el último valor de humedad leído. Se almacena como “number”.
- Temperature: almacena el último valor de temperatura leído. Se almacena como “number”.

Las acciones sobre las entidades que se pueden realizar a través de la Api están definidas en la tabla 3.

Método	URI	Descripción
GET	/v2/entities	Recupera toda la información almacenada

GET	/v2/entities/{Id}?{Atributo=Valor} &{Atributo2=Valor2}	Recupera la información almacenada que coincide con el filtro aplicado
POST	/v2/entities	Crea una nueva entidad de contexto. Se envía en el cuerpo la información relevante a la entidad
PATCH	/v2/entities/{Id}/attrs	Actualiza el valor de los atributos de la entidad de contexto. Se envía en el cuerpo un contenido similar que en la solicitud POST pero sin incluir el campo "type" e "id"
DELETE	/v2/entities/{Id}?type={type}	Elimina la entidad de contexto. En caso de haber dos entidades con mismo valor en el campo "id", tendríamos que indicar el "type".

Tabla 3: NGSI APIV2 Entidades

En la figura 63, se muestra la estructura de la entidad en formato JSON.

```
{
  "id": "Invernadero1",
  "type": "Sensor",
  "pressure": {
    "type": "Integer",
    "value": 38,
    "metadata": {}
  },
```

```
  "temperature": {
    "type": "Float",
    "value": 21,
    "metadata": {}
  }
}
```


Figura 63: Estructura de la entidad

7.2.2. Suscripciones

Las suscripciones son enviadas desde el sensor al OCB a través de la app móvil por parte del usuario.

Estas suscripciones provocan el envío de los datos de entidades de contexto por parte del OCB a Cygnus.

Vamos a proceder a explicar el funcionamiento básico de dichas suscripciones.

Las suscripciones tienen los siguientes campos:

- Id: identificador único de la suscripción.
- Status: su valor puede ser active si queremos que envíe constantemente las notificaciones (siempre y cuando se cumplan los requisitos) o oneshot que tras el envío de una notificación su estado pasa a inactive.
- Subject: determina qué tipo de entidades se actualizan en dicha suscripción. Se divide en idPattern (en nuestro caso ".*" que indica que se enviarán todas las entidades de contexto) y condition (utilizado para indicar según qué cambios se envía la notificación).
- Notification: Contiene información de la notificación:
    - timesSent: número de notificaciones enviadas desde que dicha suscripción se ha creado.
    - lastNotificaction: fecha y hora de la última notificación enviada.
    - Attrs: lista de atributos a enviar, en nuestro caso al estar vacía se envían todos los atributos.

  - http: contiene la URL a la que se envía dichas notificaciones. En nuestro caso es la dirección de Cygnus.
  - lastSuccess: fecha y hora de la última notificación enviada correctamente.
- Throttling: indica el periodo mínimo existente entre dos notificaciones.

Las acciones sobre las entidades que se pueden realizar a través de la Api están definidas en la tabla 4.

Método	URI	Descripción
GET	/v2/subscriptions	Recupera todas las suscripciones existentes
POST	/v2/ subscriptions	Crea una nueva suscripción. En el cuerpo del mensaje tiene que ir los detalles de la misma.
DELETE	/v2/subscriptions/{Id}	Elimina la suscripción cuyo id se ha indicado en el parámetro {Id}

Tabla 4: NGSI APIV2 suscripciones

En la figura 64 se muestra un ejemplo de suscripción ante cualquier cambio en cualquier tipo de entidad de contexto:

```
{
  "id": "5db50e88f2cfaec02af1fef5",
  "description": "Notify Cygnus of all context changes",
  "status": "active",
  "subject": {
    "entities": [{
      "idPattern": ".*"
    }],
    "condition": {
      "attrs": []
    }
```

```
  },
  "notification": {
    "timesSent": 2032,
    "lastNotification": "2020-01-12T04:42:44.00Z",
    "attrs": [],
    "attrsFormat": "legacy",
    "http": {
      "url": "http://CYGNUS_URL:5050/notify"
    },
    "lastSuccess": "2020-01-12T04:42:45.00Z"
  },
  "throttling": 5
}
```


Figura 64: Suscripción Cygnus

7.2.3. Funcionamiento Cygnus

Cygnus almacena toda la información que reciba en una BBDD configurada para su uso, con ello conseguimos que la información de contexto existente sea almacenada de forma persistente.

Para ello necesitamos crear una suscripción con el fin de que cada vez que se cumpla el campo subject de la suscripción el OCB emita las entidades de contexto afectadas a Cygnus y éste lo almacene en la BBDD.

La Configuración de la BBDD se indica en el fichero de configuración Docker que se adjunta en el Anexo B.

ANEXO A: INSTALACIÓN Y CONEXIÓN DEL SENSOR

Hemos desarrollado la aplicación en JavaScript, es por ello que es requisito fundamental tener instalado Node en la raspberry. En el apartado C.1 del Anexo C encontramos los pasos para la instalación de Node. Además de node, se instala su gestor de paquetes npm.

A.1 Instalación de dependencias a través de npm

A través de npm gestionamos todas las dependencias necesarias para la puesta en marcha del dispositivo BLE en la raspberry pi.

Se utilizan:

- http: Proporciona las clases y métodos para gestionar el protocolo http.
- fs: Módulo que proporciona herramientas necesarias para la administración de sistemas de archivos.
- Node-dht-sensor: Módulo encargado de leer la temperatura y humedad proveniente de sensores compatibles como: DHT22, AM2302 y DHT11.

Dichas dependencias se instalan a través de npm ejecutando el siguiente comando:

```
npm install fs http util node-dht-sensor
```

Para la puesta en marcha del dispositivo BLE ejecutamos el comando:

```
node test.js
```

A.2 Conexión Sensor AM2302 con Raspberry Pi

La conexión con el sensor se realiza directamente a los pines GPIO's de la raspberry pi. Para ello lo conectamos como se muestra en la figura 65.

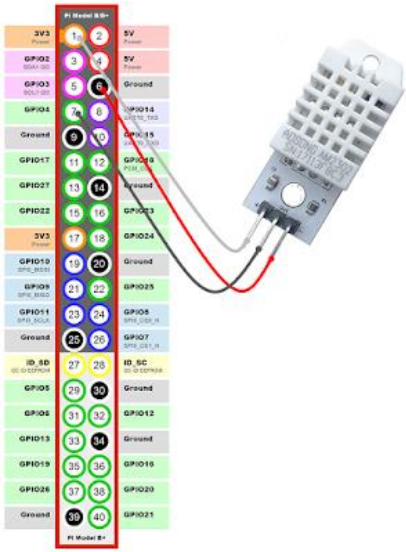

Figura 65: Conexión sensor AM2302 con RaspberryPi 3

En la figura 66 se muestra la conexión real del sensor y la raspberry pi.

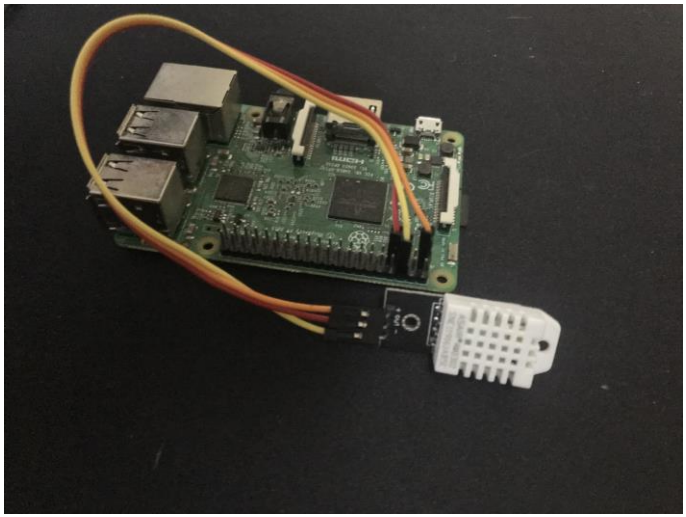

Figura 66: Conexión real sensor AM2302 con RaspberryPi 3

ANEXO B: INSTALACIÓN DE CONTEXT BROKER, CYGNUS Y MYSQL A TRAVÉS DE DOCKER

A continuación, detallamos el proceso seguido para la instalación y uso de las tecnologías Fiware utilizadas en conjunción con SQL a través de Docker.

B.1 Instalación Docker Desktop

Al utilizar Windows 10 en el ordenador que aloja la tecnología Docker, hemos procedido a descargar el instalador de Docker Desktop desde la página oficial[14].

Docker Desktop, incluye además de la tecnología "Docker Engine"; Docker CLI client, Docker Compose, Notary, Kubernetes y Credential Helper.

En nuestro caso nos interesa la tecnología Docker Compose.

Descargamos el instalador desde la referencia [https://hub.docker.com/?overlay=onboarding]. Este instalador además incluye componentes internos como el hipervisor "Hyper-V VMS" necesario para el funcionamiento Docker.

Una vez finalizada la instalación, iniciamos Docker Desktop.

Podemos comprobar la correcta instalación de las herramientas necesarias a través de los siguientes comandos:

```
docker --version
docker-compose –version
```

B.1.1. Docker-Compose

Es una herramienta que permite definir y utilizar varios contenedores Docker de manera sencilla y simultánea. Esta herramienta hace uso de los archivos de tipo YAML. En dichos archivos se definen los contenedores a utilizar así como las relaciones existentes entre ellos.

Dependiendo del contenedor podremos definir su comportamiento a través de la modificación de variables de entornos [15].

A continuación exponemos el archivo docker-compose.yaml utilizado en el proyecto explicando las partes más interesantes del documento.

B.2 Configuración de fichero Docker-Compose

La estructura del documento es la siguiente:

- Version: Indica la versión del docker-compose a utilizar.
- Services: Define cada contenedor y los describe. En nuestro caso necesitamos incluir las etiquetas:
    - Orion
    - Cygnus
    - Mysql-db
    - Mongdb
- Volumes: Etiqueta necesaria para la persistencia de los datos recolectados, mapea una dirección dentro del contenedor Docker con una del equipo en local. Permite el acceso de los datos desde la máquina contenedora.

B.2.1. Configuración OCB y Cygnus

En el primer fragmento del documento docker-compose, encontramos la definición del contenedor orion y cygnus. Además definimos el contenedor mongodb debido a que es requisito de orion para su funcionamiento interno. Se muestra en la Figura 67.

```
version: "3"

services:
  mongodb:
    image: mongo:3.4.2
    hostname: mongodb
    container_name: mongodb
    expose:
    - "27017"
    ports:
    - "27017:27017"
    command: mongod --smallfiles

  orion:
    image: fiware/orion:1.7.0
    hostname: orion
    container_name: orion
    links:
    - mongodb
    expose:
    - "1026"
    ports:
    - "1026:1026"
    command: -dbhost mongodb
cygnus:
    image: fiware/cygnus-ngsi:latest
    hostname: cygnus
    container_name: fiware-cygnus
    networks:
        - default
    depends_on:
        - mysql-db
    expose:
        - "5080"
```

```
    ports:
        - "5050:5050"
        - "5080:5080"
    environment:
        - "CYGNUS_MYSQL_HOST=mysql-db"
        - "CYGNUS_MYSQL_PORT=3306"
        - "CYGNUS_MYSQL_USER=root"
        - "CYGNUS_MYSQL_PASS=123"
        - "CYGNUS_LOG_LEVEL=DEBUG"
        - "CYGNUS_SERVICE_PORT=5050"
        - "CYGNUS_API_PORT=5080"
```

Figura 67: Creación de contenedores orion y cignus

Utilizamos las siguientes etiquetas:

- Image: define el contenedor a utilizar y la versión (:latest indica la última existente en el repositorio Docker).
- Hostname: Definimos el nombre del host del contenedor
- Container_name: Define el nombre del contenedor.
- Links: Indica dependencia entre contenedores. En este caso se observa que orion depende de mongodb.
- Command: Indica el comando que se ejecuta una vez el contenedor se está ejecutando.

Podemos comprobar que orion depende de mongodb, además orion expone los puertos 1026 al exterior y lo mapea con el puerto 1026 de la máquina local.

Además, cygnus define el puerto 5050 y 5080 (éste último con salida al exterior). Además encontramos la etiqueta environment útil para definir las variables de entorno. En este caso definimos la conexión con la BBDD MySQL utilizada por cygnus para almacenar los datos.

B.2.1. Configuración MySQL

Por último, se muestra en la figura 68 el apartado que define el contenedor MySQL.

```
mysql-db:
    restart: always
    image: mysql:5.7
    hostname: mysql-db
    container_name: db-mysql
    expose:
        - "3306"
    ports:
        - "3306:3306"
    networks:
        - default
    volumes:
        - C:/volume:/var/lib/mysql
    environment:
        - "MYSQL_ROOT_PASSWORD=123"
        - "MYSQL_ROOT_HOST=%"
```

Figura 68: Creación de contenedor MySQL

Con la etiqueta volumes, se define un espacio de memoria compartido entre el contenedor Docker y el equipo físico. El puerto utilizado por la BBDD es el 3306.

Además, definimos qué equipos remotos se pueden conectar como root (en nuestro caso indicamos cualquier host, por defecto define ”root@localhost”).

B.3. Ejecución Context Broker

Para la ejecución de los contenedores docker definidos en el documento “docker-compose.yaml” anterior, ejecutamos el primer comando (desde la carpeta que tenemos el documento docker-compose”), para pararlos ejecutamos el segundo:

```
docker-compose up
docker-compose down
```

Podemos comprobar los contenedores que están ejecutándose con el comando:

```
Docker ps
```

En la figura 69 podemos ver la salida del comando anterior.

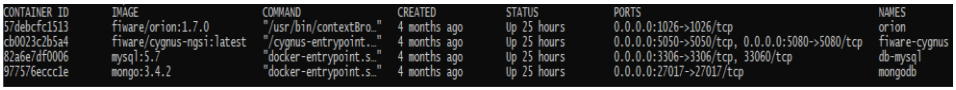

Figura 69: Ejecución contenedores Docker

ANEXO C: INSTALACIÓN DE NODE Y ANGULAR

C.1. Instalación Node

Descargamos el instalador Node para Windows desde su página web[16].

Durante el proceso de instalación instalamos el contenido adicional: "npm package manager" y "Add to PATH" además de "Node.js runtime".

Una vez finalizado el proceso de instalación, tendremos el entorno de ejecución de JavaScript instalado, además podremos usar la herramienta "npm" para la gestión de dependencias.

Desde este momento podemos utilizar el comando node para ejecutar cualquier fichero JavaScript.

C.2. Instalación y uso de Angular-cli

Gracias a npm podemos instalar de forma cómoda Angular, también nos resulta de gran utilidad para la instalación de Ionic.

Para la instalación de Angular y desde cualquier ventana de comandos, ejecutamos:

```
npm install angular
```

Angular-cli es una potente herramienta que permite la generación de proyectos de angular de forma sencilla, así como de componentes individuales en dichos proyectos, modificando además las referencias internas para facilitar el trabajo del programador.

En la tabla 5 recopilamos los comandos más importantes usados durante el desarrollo del proyecto:

`npm install @angular/cli`	Instalación Angular-cli

`ng new nombre_proyecto`	Creación del directorio básico para proyectos Angular.
`ng serve`	Pone en marcha el proyecto
`ng generate component Nombre_componente`	Genera un nuevo componente angular definiendo nuevas referencias dentro del proyecto.

Tabla 5: Comandos angular-cli

C.2. Creación del proyecto Node y tips del programador.

Para la creación de un proyecto node ejecutamos el comando:

```
npm init
```

Ejecutará un asistente de inicialización y finaliza con la creación de un fichero package.json.

Dicho documento almacena información general del proyecto así como las dependencias. Su existencia permite descargar de manera sencilla dichas dependencias a través del comando:

```
npm install
```

Una vez obtenido el fichero package.json, procederemos a incluir la etiqueta build y dev. Se muestra en la Figura 70.

```
"scripts": {
    "build": "tsc -w",
    "dev": "nodemon build/index.js"
  },
```

Figura 70: Etique buid y dev

Esto añade dos scripts que podemos ejecutar con el comando:

```
npm run <<script>>
```

El script "build" ejecuta el compilador de TypeScript, transformando el código TypeScript en JavaScript, el segundo, comprueba modificaciones en el código para reiniciar su ejecución.

Ambos scripts resultan de gran utilidad durante el proceso de desarrollo.

C.3. Instalación de Express y Promise-MYSQL

Como se ha indicado en el apartado 6.1. Módulos Servicio Web, ambos módulos se encuentran incorporados en el gestor de paquetes Node, por tanto su intalación se realiza mediante npm:

```
npm install express promise-mysql
```

ANEXO D: INSTALACIÓN IONIC FRAMEWORK

La instalación del framework ionic se realiza mediante el gestor de paquetes de Node en el equipo donde se quiere desarrollar la aplicación móvil. Para su instalación se puede consultar el Anexo C. Los requisitos del sistemas son similares a los necesarios para ejecutar javascript en Node, en nuestro caso lo instalamos para Windows, estando disponible en diferentes distribuciones Linux y MacOS.

Para su instalación se ejecuta por línea de comandos en cualquier terminal:

```
npm install cordova ionic@3
```

Una vez finalizada la instalación ejecutaremos el asistente de creación de proyectos ionic a través del comando:

```
Ionic start
```

Esto ejecutará un asistente de generación de proyectos Ionic, creando la estructura del proyecto. Además nos da la posibilidad de elegir entre:

- Nombre del proyecto.
- Plantilla de proyecto a elegir: Diferentes plantillas orientadas a productos finales. Según la finalidad de la aplicación podremos usar una plantilla y otra. Nos permite además, incluir plantillas subidas en respositorio git.
- Versión de ionic a utilizar.
- Incluir o no cordova en la instalación.

Nos generará la estructura del proyecto siguiendo la estructura de componentes de Angular, además nos da la posibilidad de elegir entre diferentes plantillas (en nuestro caso elegimos blank, generando un proyecto vacío).

Una vez finalizado los trabajos de desarrollo, se puede generar la aplicación e instalar en un dispositivo android conectado mediante usb a través de:

```
Ionic cordova run android
```

ANEXO E: DEPURACIÓN

D.1. Depuración REST API

Para la depuración de la API REST que hemos desarrollado hemos utilizado la herramienta Postman.

Se descarga desde la página web de Postman[17].

Podemos realizar peticiones HTTP con los diferentes métodos existentes en el protocolo pudiendo añadir cabeceras, el cuerpo del mensaje o incluso añadir un token a la petición para poder acceder a un sistema que funciona por ejemplo con Java Web Tokens.

En la figura 71 podemos observar la respuesta del OCB a la petición GET que obtiene las entidades existentes en el sistema.

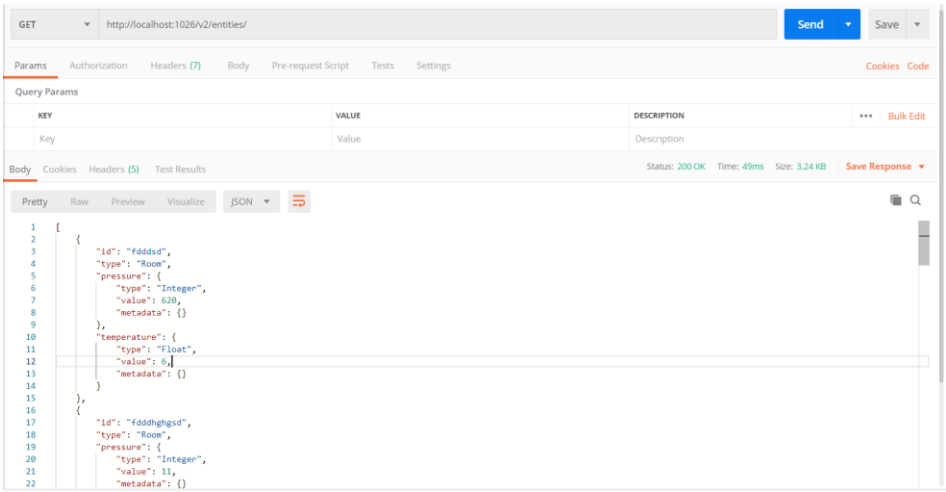

Figura 71: Envío petición GET al OCB

D.2. Depuración APP Móvil

Para la depuración de la App hemos usado la herramienta incluída en Android Studio. La app se ha desarrollado a través del framework Ionic. Este framework genera una apk del código desarrollado en Angular con el fin de ser utilizado en un dispositivo móvil.

En la Figura 72 encontramos la ubicación de la apk, para su posterior depuración de la aplicación instalada en el dispositivo móvil conectado por usb.

```
BUILD SUCCESSFUL in 31s
44 actionable tasks: 2 executed, 42 up-to-date
Built the following apk(s):
        C:\BLE_APP_V3\platforms\android\app\build\outputs\apk\debug\app-debug.apk

ANDROID_SDK_ROOT=undefined (recommended setting)
ANDROID_HOME=C:\Users\evall\AppData\Local\Android\Sdk\ (DEPRECATED)
No target specified, deploying to device 'CQ3000EK31'.

Using apk: C:\BLE_APP_V3\platforms\android\app\build\outputs\apk\debug\app-debug.apk

Package name: io.ionic.starter
LAUNCH SUCCESS

[OK] Your app has been deployed.
     Did you know you can live-reload changes from your app with --livereload?

PS C:\BLE_APP_V3> _
```

Figura 72: Ubicación apk

Para empezar la depuración tenemos que abrir la pestaña “File” de Android Studio y pinchamos en “Profile or debug APK”.

Una vez iniciada la aplicación en el dispositivo móvil, podremos visualizar los logs en tiempo de ejecución desde Android Studio.

D.3. Depuración BLE

Para la depuración de código en relación a la comunicación entre el dispositivo BLE y la aplicación móvil hemos usado la aplicación LightBlue.

La propia instalación se realiza desde la App Store de IOS o desde la Play Store de Android.

Nos permite estudiar los servicios y características de los periféricos BLE. Además nos facilita una interfaz de usuario simple para interactuar con ellos. Esta aplicación convierte al teléfono móvil en un servidor en la conexión BLE, siendo el sensor BLE el cliente.

En la figura 73 podemos ver el uso de la aplicación una vez nos hemos conectado a un dispositivo BLE. Podemos obtener los datos enviados a difusión así como los servicios que tiene disponible.

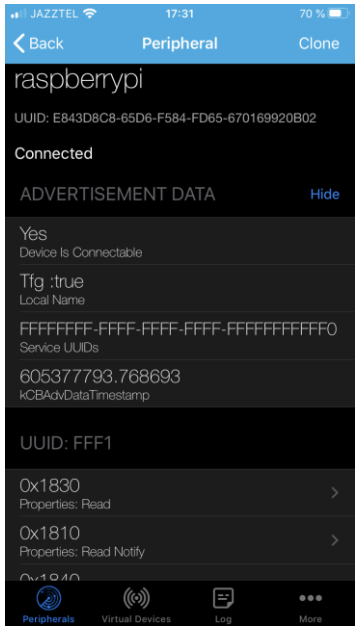

Figura 73: Conexión con sensor BLE a través de LightBlue

Además en la Figura 74 observamos que podemos hacer uso de dichos servicios sin necesidad de haber desarrollado la aplicación móvil para comprobar el funcionamiento del sensor BLE.

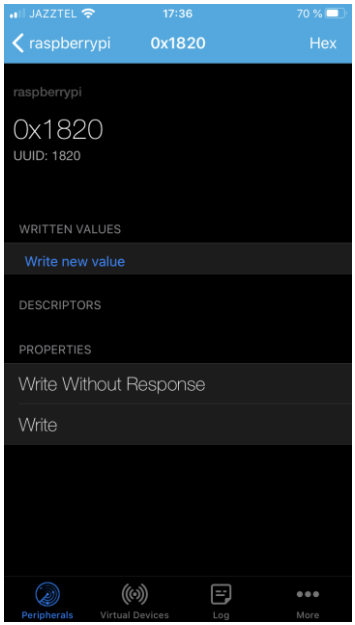

Figura 74: Servicio cambio de nombre a través de LightBlue.

ANEXO F: PUESTA EN MARCHA.

La puesta en marcha del sistema se realiza mediante un script desarrollado en Python, que ofrece un menú básico. La figura 75 muestra el menú de arranque.

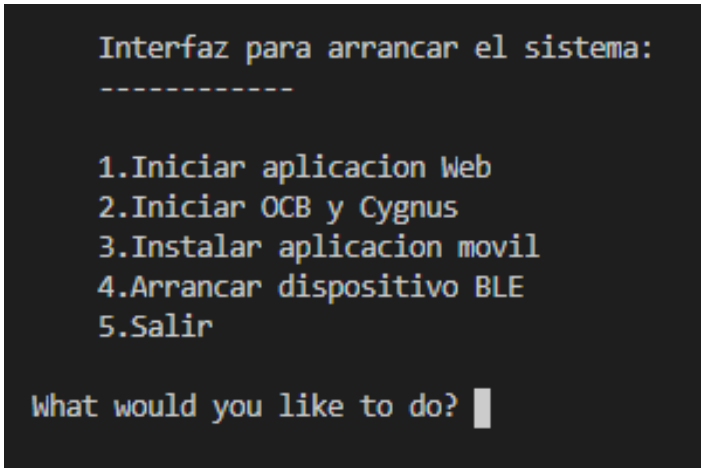

Figura 75: Menú principal script arranque.

F.1. Arranque Aplicación web.

Al ejecutar el script de arranque y seleccionar la opción 1, se abren tres terminales Windows dónde se arranca por un lado el proyecto Angular, y en las dos restantes se arranca el servidor Node. El propio menú indica que para apagar la aplicación web hay que cerrar las terminales. Se ejecutan los siguientes comandos:

```
cd Aplicacion_Web/Cliente && ng serve
cd Aplicacion_Web/server && npm run build
cd Aplicacion_Web/server && npm run dev
```

F.2. Arranque de OCB y Cygnus a través de Docker.

Al ejecutar el script de arranque y seleccionar la opción 2, se abre una terminal que ejecuta el comando.

```
cd OCB_Docker && docker-compose up
```

La sencillez para arrancar tanto OCB y Cygnus, como MySQL y MongoDB, se debe a la tecnología docker-compose de Docker.

Este comando lee el fichero docker-compose.yml ubicado en la carpeta OCB_Docker del proyecto y ejecuta las instancias Docker definidas en él.

F.3. Instalar aplicación móvil.

Al seleccionar la opción 3 del script de arranque, se procede a la instalación de la aplicación móvil en un sistema Android.

Para ello se utiliza el framework Ionic y Cordova, y basta tener enchufado el móvil al ordenador por USB. El proceso genera una APK y lo transfiere al dispositivo enchufado.

Se muestra a continuación el comando ejecutado.

```
cd Aplicacion_Movil && ionic cordova run android
```

El inicio de la App se hace desde el propio dispositivo, y el usuario tiene que darle permisos de ubicación.

F.4. Arrancar Dispositivo BLE

Una vez seleccionada la opción que procede a arrancar el dispositivo BLE, le tenemos que indicar su dirección IP.

Debido a que el sistema operativo se ubica en la tarjeta SD introducida en la raspberry pi, no es necesario transferir el proyecto.

Únicamente se procede a conectarse a través de ssh al dispositivo y ejecutar el proyecto. En la figura 76, se muestra el diálogo de arranque.

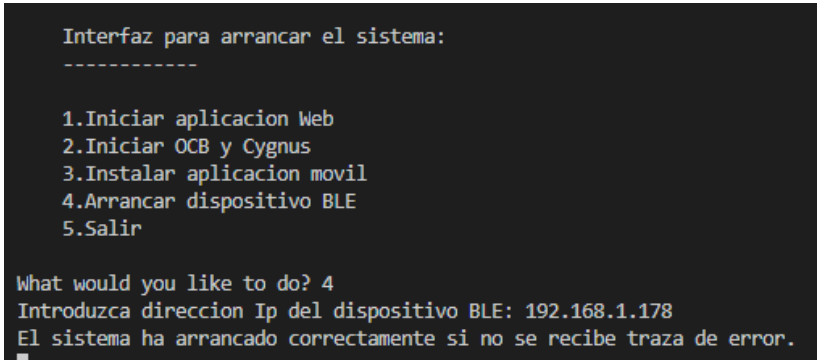

Figura 76: Arranque dispositivo BLE.

El comando que ejecuta en el dispositivo se muestra a continuación.

```
cd /home/pi/dht/node_modules/bleno && sudo node test.js
```

REFERENCIAS

[1] MICROSOFT. VISUAL STUDIO CODE: GETTING STARTED. [CONSULTA: 14 DE FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://CODE.VISUALSTUDIO.COM/DOCS

[2] ANDROID STUDIO. WIKIPEDIA: LA ENCICLOPEDIA LIBRE.[CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://ES.WIKIPEDIA.ORG/WIKI/ANDROID_STUDIO

[3] ORACLE CORPORATION. MYSQL WORKBENCH. ENHANCED DATA MIGRATION. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://WWW.MYSQL.COM/PRODUCTS/WORKBENCH

[4] ANGULAR (FRAMEWORK). WIKIPEDIA: LA ENCICLOPEDIA LIBRE.[CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://ES.WIKIPEDIA.ORG/WIKI/ANGULAR_(FRAMEWORK)

[5] BOOTSTRAP TEAM BOOTSTRAP. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://GETBOOTSTRAP.COM

[6] OPEN JS FOUNDATION. NODE JS ([CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://NODEJS.ORG/ES/

[7] IONIC. IONIC FRAMEWORK 5. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://IONICFRAMEWORK.COM

[8] DOCKER DOKERCON LIVE 2020.. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://WWW.DOCKER.COM

[9] MINISTERIO DE CIENCIA E INNOVACIÓN (ESPAÑA). PROYECTO FI-WARE. 27 DE AGOSTO DE 2013 [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: DE HTTPS://WWW.REDIRIS.ES/PROYECTOS/FI-WARE/INDEX.HTML.ES

[10] SOLID GEAR PROJECTS.BLUETOOTH BLE: EL CONOCIDO DESCONOCIDO. [CONSULTA: FEBRERO 2020]. DISPONIBLE EN: HTTPS://SOLIDGEARGROUP.COM/BLUETOOTH-BLE-EL-CONOCIDO-DESCONOCIDO/?LANG=ES

[11] MKDOCS.. FIWARE: APRENDE FIWARE EN ESPAÑOL [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://FIWARE-TRAINING.READTHEDOCS.IO/ES_MX/LATEST/ECOSISTEMAFIWARE/PLATAFORMAFI

WARE/
[12] ADAFRUIT. INTRODUCTION TO BLUETOOTH LOW ENERGY GATT. 20 DE MARZO DE 2014 [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://LEARN.ADAFRUIT.COM/INTRODUCTION-TO-BLUETOOTH-LOW-ENERGY/GATT
[13] EXPRESS FUNDACIÓN NODE.JS. INFRAESTRUCTURA WEB, RÁPIDA MINIMALISTA Y FLEXIBLE PARA NODE.JS. [CONSULTA: 18 DE FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://EXPRESSJS.COM/ES
[14] DOCKER. DOCKER [PROGRAMA]. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://DOCS.DOCKER.COM/DOCKER-FOR-WINDOWS/INSTALL/
[15] DOCKER. OVERVIEW OF DOCKER COMPOSE. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://DOCS.DOCKER.COM/COMPOSE/
[16] OPEN JS FOUNDATION. NODEJS [PROGRAMA]. [CONSULTA: FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://NODEJS.ORG/ES/DOWNLOAD/
[17] POSTMAN. POSTMAN [PROGRAMA]. [CONSULTA: 14 DE FEBRERO DE 2020]. DISPONIBLE EN: HTTPS://WWW.POSTMAN.COM/DOWNLOADS/